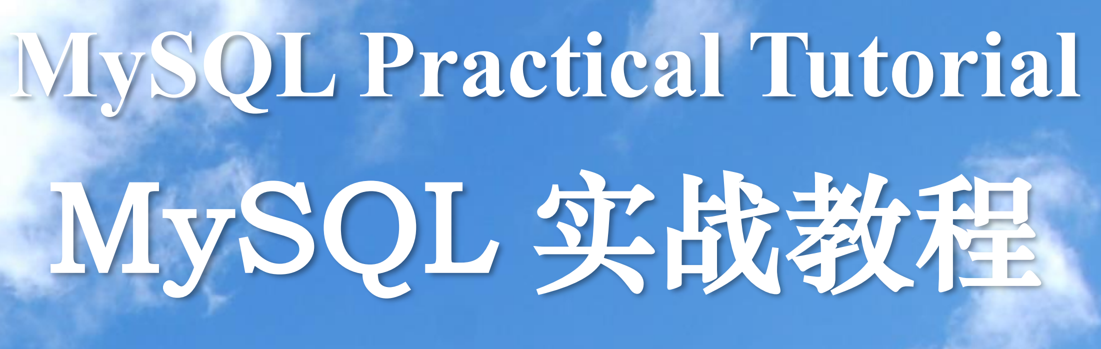


（电子书 ebook）

黄能耿 胡丽丹 主编

v1.0.0

两大特色

40 余个 Jitor 在线实训

10 余个项目案例 ＋ 项目网站

版权所有 翻印必究 欢迎传播

2024 年 7 月 23 日

Copyright statement 版权声明

This PDF document (referred as this book) is published in the form of an e-book. This book can be freely copied and distributed for non-commercial purposes and without any modification. Any modification of this book, especially modification of the signature and modification of this copyright statement, is not allowed. Commercial dissemination or printing of hard copies of this book is not allowed. Converting this book to other electronic formats or adding or deleting it is also not allowed.

You are welcome to use and distribute this book. This book can be used for any learning and teaching purposes, including teaching at all levels of schools and training institutions. The latest version of this book, related resources and software can be downloaded from the homepage of this book http://ngweb.org/mysql/ where you can dowload the "Jitor Online Lab" and "Project Development Platform on Browser" that come with this book are developed by the author of this book and are completely free to use.

The author reserves all copyrights of this book and hereby declares.

本 PDF 文档（以下简称本书）以电子书的形式发布，本书可以在非商业用途和未作任何修改的前提下被自由复制和传播，对本书的任何修改，特别是对署名的修改以及对本版权声明的修改都是不被允许的，对本书的商业传播或印刷为纸质副本都是不被允许的，将本书转换为其他电子格式或进行增加或删减也是不被允许的。

欢迎使用和传播本书，本书可用于任何学习和教学用途，包括各级学校的教学和培训机构的培训。本书的最新版本、相关资源和软件可从本书主页 http://ngweb.org/mysql/下载，本书配套的“Jitor 校验器”和“实战演练平台”由本书作者开发，完全免费使用。

作者保留本书的全部版权，特此声明。

Introduction 内容提要

The book "MySQL Practical Tutorial" is a textbook, used in introduction to the basic knowledge and development techniques of MySQL database. This book emphasizes practicality, takes multiple practical projects as the main line, and arranges relevant knowledge and skills reasonably. This book covers relational database theory and design, basic database operations (DDL, DML, DQL), subqueries, views, indexes, database programming and database management, combines knowledge and skills into practical projects, explains some practical skills (views, transactions, query skills, performance optimization and indexing, security skills) and the development of practical projects. The book ends with a demonstration project of a PSIMS (Purchase, Sales and Inventory Management System).

This book provides more than 40 Jitor trainings on "Jitor Online Lab" and more than 10 practical projects (project cases + project websites) on "Project Development Platform on Browser". The Jitor training can evaluate the code written by students and the results of the operation online, and teachers can review the progress of the practical training of all students online. The practical project adopts the teaching mode of "project case + project website". The SQL code written by readers in the project case can interact with the project website. Not only can they learn and develop on the MySQL client end like MySQL Workbench, but they can also write SQL statements on "Project Development Platform on Browser" (such as Chrome browser) for learning and development any projects with full functions without to write any codes other than SQL statements. These are the major two features of this book.

This book is an introductory textbook for database-related courses. It is suitable for students and self-learners in beginner level of all readers.

本书讲解 MySQL 数据库的基础知识和开发技术。本书突出实用性，以多个实战项目为主线，合理安排相关知识点和技能点。本书涵盖了关系数据库理论和设计、数据库的基本操作（数据定义、数据操纵和数据查询），子查询、视图、索引、数据库编程和数据库管理，将知识点和技能点结合到实战项目中，讲解一些实战技巧（视图的应用、事务的应用、查询技巧、性能优化与索引、安全技巧）以及实战项目的开发，并提供了一个进销存管理系统的综合性演示项目。

本书设计了 40 余个 Jitor 在线实训和 10 余个实战项目（项目案例+项目网站），每个单元至少有 1 个实战项目。Jitor 实训可以对学生编写的代码和运行的结果进行实时评价，教师可以实时监测全班学生的实训进展情况。实战项目采用“项目案例+项目网站”的教学模式，读者在项目案例上编写的 SQL 代码都可以与项目网站互动，不仅可以在 MySQL 客户端上进行学习和开发，也可以在项目网站（浏览器）上编写 SQL 语句进行学习和开发，这是本书的最大特色。

本书是数据库相关课程的入门教材，适用于广大自学者和在校学生，适用于计算机相关专业和非计算机专业，可以作为高等职业院校、中等职业院校、培训机构的教材，可以作为全国计算机等级考试二级 MySQL 数据库程序设计的参考用书，也可作为应用型本科实验教材。

Title 书名：MySQL Practical Tutorial; MySQL 实战教程

Author 作者：Nenggeng Huang, 黄能耿; Lidan Hu, 胡丽丹

Website 网址：http://ngweb.org/mysql/

Language 语言：Chinese, 中文

Version v1.0.0 July 23th. 2024; 第 1 版 v1.0.0 2024 年 7 月 23 日

Copyright © 2024~2024 by 黄能耿 胡丽丹

前言

完成了本书的写作之后，但却被告知还需要漫长的等待才能出版，心中不免有些郁闷。原本是作为教材而编写的，特别是书中还有两项独一无二的特色，寄冀于本书能够尽快到达读者的手中，能够让更多的读者阅读，因此就有了这个电子版，幸好还未与出版社签订出版合同，可以自由支配本书的版权，同时也希望读者能够喜欢这本书。

数据库技术是信息管理的关键技术，它能够高效地存储、检索、更新和管理大量的数据，是各行业不可或缺的基础设施，是计算机相关专业最为核心的课程之一。本书遵循高职学生的认知和技能形成规律，重点突出、通俗易懂、编排合理，以关系数据库的典型代表 MySQL 为例，讲解关系数据库技术的基础知识。本书单元 1~单元 5 是基础部分，涵盖了关系数据库理论和设计、数据库的基本操作（数据定义、数据操纵和数据查询），单元 6~单元 9 侧重于数据库开发，涵盖了子查询、视图、索引、数据库编程和数据库管理，讲解了一些实战技巧（视图的应用、事务的应用、查询技巧、性能优化与索引、安全技巧）以及实战项目的开发，并提供了一个进销存管理系统的综合性演示项目。

本书采用 2024 年发布的 MySQL 8.0.36 版，MySQL 客户端采用 MySQL 官方提供的 MySQL Workbench，数据库建模工具也使用 MySQL Workbench。

本书的两个特色是提供了 40 余个 **Jitor 在线实训**和 10 余个**实战项目（项目案例+项目网站）**，每个单元至少有 1 个实战项目。Jitor 实训可以对学生编写的代码和运行的结果进行实时评价，教师可以实时监测全班学生的实训进展情况。实战项目采用“项目案例+项目网站”的教学模式，读者在项目案例上编写的 SQL 代码都可以与项目网站互动。在实战项目中，读者只需 SQL 语句就可以开发出完整的 B/S 应用项目，从而极大的提高学生的学习兴趣，符合高职高专教育的特点。

本书以实战项目驱动并贯穿全部教学内容，从内容安排、知识点和技能点的组织、教与学、做与练、技能学习与项目开发等多方面体现高职教育特色。本书的主要特点体现在以下三个方面。

1. 项目贯穿、案例驱动

本书通过 10 余个实战项目（项目案例+项目网站），以完成项目所需的知识和技能点来组织教学内容，并通过实战项目来实际运用所学的知识和技能，从而达到巩固学习效果的目的。

2. 以岗定课、课岗直通

本书以数据库开发的岗位需求来选择教学内容，选用主流技术，介绍新的开发理念，帮助读者做到无缝衔接课堂所学与岗位需求。

3. 产教融合、校企合作

本书采用的“项目案例+项目网站”教学模式是产教融合的产物。编者借鉴企业的真实项目开发了用于 MySQL 教学的“实训演练平台”。本书最后一个单元也参考了企业真实项目和员工培训的经验，是根据企业的实际需求而提炼出来的。

本书各单元的特色内容与素质培养对照见表 1。

表 1 特色内容与素质培养对照

单元	特色内容	素质培养
单元 1 认识数据库	通过“Hello，数据库”实例引入主键的概念	服务社会、社会责任感
单元 2 理解关系数据库	通过“图书信息数据库”实例引入外键的概念	逻辑思维能力、万物皆有联系

单元 3 数据定义	数据完整性约束，数据建模工具	实体间的联系、全局观念、道德规范
单元 4 数据操纵	增删改与数据完整性约束的关系	规则的重要性、遵守法律法规
单元 5 数据查询	Where 条件查询、连接查询	保护用户隐私、防止数据泄露、专业道德
单元 6 子查询、视图和索引	子查询与增删改、视图、索引	逻辑思维能力、嵌套、递归、抽象
单元 7 数据库编程	事务和锁	多用户环境、团队精神、诚信与责任
单元 8 数据库管理	MySQL 命令行、数据库安全、数据备份和恢复	安全意识、诚信与法治
单元 9 项目实战	实战技巧，视图和事务的应用，进销存管理系统	创新思维、批判性思维、终身学习

本书作为自学使用时，建议下载“Jitor 校验器”并注册一个账号，在 Jitor 实训的指导下完成每一个实训任务。在学完“单元 5 数据查询”后，可以先学习“单元 9 项目实战”的“9.2 图书信息项目的开发”，再接着学习“单元 6 子查询、视图和索引”。

本书作为教学使用时，参考课时为 48~64 课时，建议采用理实一体化教学模式，充分利用 Jitor 实训和实战项目资源。各单元的参考课时见表 2，在本书配套的授课计划中有详细说明，授课计划可从本书主页 http://ngweb.org/mysql/?p=8 下载。

表 2 课时安排建议

单元		课时建议
基础部分 数据库的基础 应该循序渐进地学习	单元 1 认识数据库	2～4
	单元 2 理解关系数据库	8～10
	单元 3 数据定义	6～10
	单元 4 数据操纵	2～4
	单元 5 数据查询	10～12
提高部分 多数二级标题可以独立成篇 可以选择性地学习	单元 6 子查询、视图和索引	4～8
	单元 7 数据库编程	4～8
	单元 8 数据库管理	4～6
	单元 9 项目实战	6～或专用周
合计		46～62

本书提供教学大纲、教学整体设计、授课计划、教案、PPT 课件、习题答案、全书源代码等资源，可从 http://ngweb.org/mysql/?p=8 下载（见附录 E）。

本书由无锡职业技术学院的黄能耿、胡丽丹担任主编，Jitor 实训教学平台、Jitor 实训和实战项目由黄能耿研发和编写，感谢编者所在单位领导和同事的帮助和大力支持。

由于编者水平所限，书中疏漏和不足之处在所难免，敬请广大读者批评指正。

编者

2024 年 7 月

目录

【基础篇】

本书的单元 1~单元 5 是基础篇，请读者循序渐进地学好每一单元的内容，特别是要学好单元 1 和单元 2 的内容，打好关系数据库的基础。

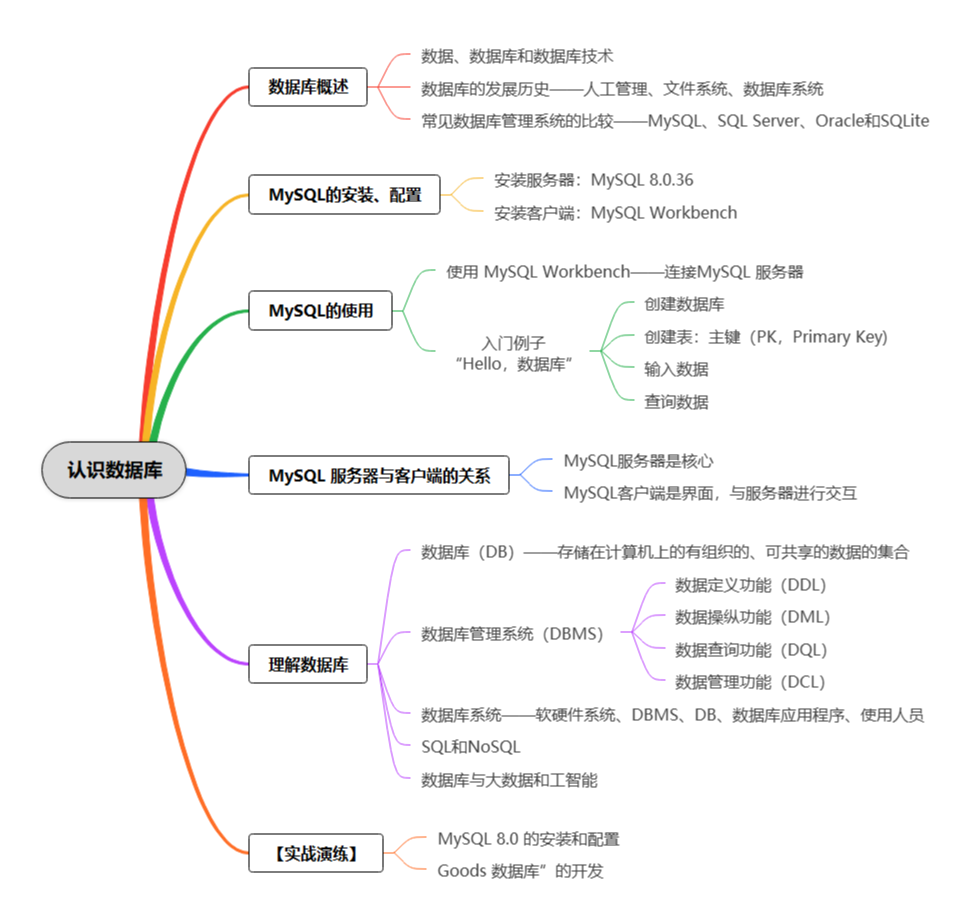
每个单元都有一些实训和实战项目，请下载“Jitor 校验器”（http://ngweb.org/mysql/?p=8），学生需要由教师为其创建账号，普通读者可以自行注册免费账号，在实训指导下完成每一步操作，“实战演练平台”是集成在“Jitor 校验器”中的，登录后就可体验实战项目，使用方法详见附录 C 和附录 D。

单元1 认识数据库

【学习目标】

知识目标	能力目标
◆ 了解数据库的基础知识。	◆ 掌握 MySQL 的安装和配置方法。
◆ 了解 4 种常用的数据库管理系统。	◆ 熟练掌握 MySQL Workbench 的基本使用方法。
◆ 理解主键的概念、地位和作用。	**素质目标**
◆ 理解数据库管理系统及其功能。	◆ 提高学生对专业学习的认可度与专注度。
◆ 了解关系数据库和非关系数据库。	◆ 培养学生服务社会的理念，增强社会责任感。

【思维导图】



【情景导入】

软件技术专业的学生小明听了入学后的专业介绍，并从网上查阅了一些资料之后，知道计算机相关专业的学生都必须学习数据库这门课程，甚至连非计算机相关专业的学生也有学习数据库课程的，因为在流量为王的时代，数据已经变得越来越重要。他又从学长那里得知，学习数据库入门的首选是学习 MySQL，最好是通过项目案例的开发来学习，边学习边开发，这样效果最好，毕业后到了开发岗位马上就能学有所用，服务于社会，做一个有社会责任感的人。

【知识储备】

计算机技术研究的对象分为两大部分，软件和数据。软件是用计算机语言编写的，这些软件可以是传统的软件，可以是互联网上的网站，也可以是手机上的 APP。数据是软件处理的对象，数据可能很少，也可能很多，数据可以分散保存，也可以集中保存，数据可能是独享的，也可能需要通过网络共享。

在大数据和人工智能时代，各种各样的数据越来越多，数据也与我们的日常生活密切相关，手机 APP 里就拥有大量的数据。数据已经渗透到了各行各业，从应用行业来看，从工业、农业、服务业，到金融业、科研等，从应用类型来看，从传统的数据分析、数据建模，到人工智能、大数据处理，数据已经是无处不在，无时不有，数据将变得越来越重要。

1.1 数据库概述

数据库技术是对数据进行整理、处理和分析的技术，是软件技术专业、大数据技术专业以及计算机相关专业最为核心的课程之一。

1.1.1 数据和数据库

1. 数据

数据（Data）是对客观事物的描述。在计算机中，数据是以可识别的符号表现的，这些符号可以是数字，也可以是文字、图像、音频、视频等。

数据随处可见，例如用手机拍摄的照片和视频，记录在手机里的数字和文字。

2. 数据库

数据库（Database，DB）是存放数据的仓库。在计算机中，数据库是存储在计算机上的有组织的、可共享的数据的集合。为了有效地存储和利用数据，要求这些数据以一定的方式储存在一起、能为多个用户共享、具有尽可能小的冗余度，是与应用程序彼此独立的数据集合。

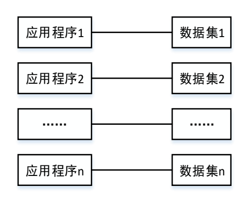
数据和数据库之间存在着密切的关系。简单来说，数据是数据库存在的基础和核心内容，而数据库则是数据的容器和管理体系。

3. 数据库技术

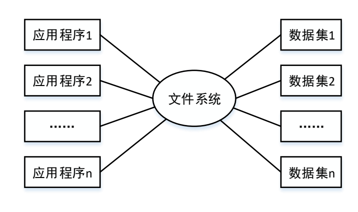
数据库技术研究如何科学地组织和存储数据，如何有效地获取和处理数据，如何高效地进行数据的查询、检索、更新和管理，如何保证数据的完整性、有效性和安全性，并将数据的处理和管理从应用软件中独立出来，形成一门独立的技术。

1.1.2 数据库的发展历史

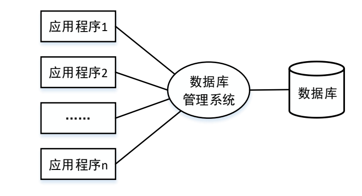
随着计算机技术的发展，数据处理随之出现，早期的数据处理是依附于软件的，后来才形成了独立的数据处理技术。因此，数据库的发展经历了三个阶段，如图 1-1 所示。



a) 人工管理阶段



b) 文件系统阶段



c) 数据库系统阶段

图 1-1 数据库发展的三个阶段

1. 人工管理阶段

20 世纪 50 年代，计算机技术还处于早期发展阶段，数据是直接由应用软件管理的，数据与应用软件紧密梆定。

在这个阶段，没有专门用于管理数据的软件，每种应用软件都要自行负责数据的处理和管理，因此加大了数据处理的难度和复杂度，如图 1-1 a）所示。

2. 文件系统阶段

20 世纪 60 年代前后，随着计算机技术的发展，以及软硬件性能的提升，出现了专门管理数据的软件，这种软件采用文件系统来处理和管理数据，数据有了一定程度的独立性，可以与应用软件分开。

由于文件系统功能的局限性，无法有效地实现数据操纵和数据查询，也不能反映现实世界里各种事物之间复杂的关系，如图 1-1 b）所示。

3. 数据库系统阶段

20 世纪 60 年代开始，开始出现了通用的数据库技术，其特点是数据独立于应用系统，如图 1-1 c）所示。在这个阶段，先后出现了四种类型的数据库技术。

- 层次数据库：采用层次模型作为数据的组织方式，层次模型是一种树状结构，其特点是每个节点最多只有一个父节点，在描述现实世界时受到诸多限制，目前已经被淘汰。
- 网状数据库：采用网状模型作为数据的组织方式，网状模型是由树状结构发展而来，网状模型更加灵活，其特点是一个节点可以有多个父节点，可以形成网状联系，但是也已经被淘汰。
- 关系数据库：采用关系模型作为数据的组织方式，这是数据库技术的主流，本书只讲解关系数据库。
- 面向对象数据库：采用对象技术来存储数据，它不仅是数据库系统，同时也是面向对象的系统。虽然该技术的理念非常先进，但是还不够成熟，还无法与关系数据库技术抗衡。

1.1.3 数据库引擎排行榜

MySQL 是一种数据库管理系统，其核心是数据库引擎。数据库引擎排行榜是业内知名网站 DB-Engines 根据实时收集的使用情况、市场占有率等数据给出的数据库引擎的人气排名，表 1-1 列出了 2024 年 2 月排名前十的数据库引擎，这些数据库分为两大类：关系数据库和非关系数据库。

表 1-1 数据库引擎排行榜（2024 年 2 月）

排行	数据库引擎	分类	说明
1	Oracle	关系数据库	经典的数据库，广泛应用于大型企业（如银行等）
2	MySQL	关系数据库	开源的数据库，被 Oracle 公司收购，广泛应用于中小型应用，甚至是大型应用
3	SQL Server	关系数据库	Microsoft 公司的产品，特点是容易学习，广泛应用于中型企业
4	PostgreSQL	关系数据库	同时也是一种面向对象数据库，在中小型企业都有应用
5	MongoDB	非关系数据库	文档存储数据库
6	Redis	非关系数据库	键值（Key-Value）存储数据库
7	Elasticsearch	非关系数据库	全文搜索引擎数据库
8	IBM DB2	关系数据库	老牌的数据库，在遗留的旧系统中仍有广泛应用
9	Snowflake	关系数据库	完全基于云的一种数据库
10	SQLite	关系数据库	适合嵌入式应用，使用非常广泛，是安卓手机的内置数据库

从表 1-1 可以看出，MySQL 在数据库引擎排行榜中排名第二，并且前三名的名次自 2013 年 9 月以

来就没有改变过。MySQL 是一种应用广泛的数据库，适合中小型应用。在网站建设中，MySQL 的应用高居第一，达到近 70%的市场份额。

学习数据库的入门首选是 MySQL 或 SQL Server，而 Oracle 则过于庞大和复杂，不适合初学者学习。其他几种数据库管理系统要么是不够普及，要么是过于简单，要么是属于非关系数据库，都不适合初学者学习。

本书选择 MySQL 是由于它的普及面广、功能丰富、难易适中，特别是长期以来，市场上对 MySQL 的需求旺盛，还有许多国产数据库管理系统是基于 MySQL 的源码进行二次开发而成的，因此应用前景非常好。另外，关系数据库都是基于通用的 SQL 标准，学会了 MySQL，基本上就可以无师自通地学习其他几种关系数据库，例如 SQL Server、Oracle 和 SQLite 等。

1.1.4 数据库管理系统的比较

表 1-1 列出了排名前 10 的数据库管理系统，常见的数据库管理系统有 Oracle、MySQL、SQL Server 和 SQLite，这 4 种数据库管理系统以及非关系数据库的比较如表 1-2 所示。

表 1-2 常见数据库管理系统的比较

数据库引擎	比喻为交通工具	软件大小	安全性	价格	数据量	典型用户
Oracle	喷气式客机	几个 GB	高	数百万元	亿级	所有银行
SQL Server	客运火车	几个 GB	较高	数万元	千万级	大中型企业
MySQL	小轿车	几百 MB	较高	社区版免费	百万级	大中小型企业、网站
SQLite	自行车	几个 MB	低	免费	万级	个人（手机、普通计算机）
非关系数据库	货运交通工具	种类繁多，各不相同，有不同的指标，难以比较				

如果将数据库比作运输工具，如表 1-2 所示，那么关系数据库是载客类的交通工具，非关系数据库是货运类交通工具。Oracle 是喷气式客机，非常大，安全性极高；SQL Server 是火车，也非常大，安全性很高；MySQL 是小轿车，小而安全，可以普及到家庭；SQLite 是自行车，小而简单，可以免费使用。非关系数据库是各种各样的货运交通工具，不同的非关系数据库适合处理不同的数据，以满足不同的需求。

如果从产品的存储空间大小和价格来比较，Oracle 和 SQL Server 软件大约几个 GB，只有收费的商业版，价格是几万到几十万元，甚至几百万元一套；MySQL 软件大约几百 MB，收费的商业版提供技术支持，而免费的社区版的功能与商业版的功能完全相同，可以在生产中使用，唯一的区别是不提供技术支持；SQLite 软件大约是几 MB，并且是完全免费的，每一台安卓手机都预安装了 SQLite，许多 Windows 或 Linux 应用软件也将 SQLite 作为一个组件，而嵌入在产品中，所以每台计算机上都安装了多个 SQLite。

如果从管理的数据来看，Oracle 可以管理一家大型银行，包括几千万至几亿储户的存款和贷款记录，每天可以有数百万、上千万条数据的变化；MySQL 可以管理一个网站数万到数十万的数据变化；SQLite 可以管理手机上数千条数据变化；非关系数据库管理的则是其他类型的数据，例如文档存储数据或键值存储数据等。

再从全球的应用来看，使用 Oracle 和 SQL Server 的只有数万家到几十万家大中型企业；使用 MySQL 的有数以千万计的网站，以及数以千万计的应用软件用户；使用 SQLite 的有数以十亿计的安卓手机用户，以及数以亿计的应用软件用户；使用各种非关系数据库的则是数以百万计的有各种特殊应用需求的用户。

1.2 MySQL 的安装和配置

MySQL 是一种基于 SQL 标准的关系数据库管理系统，从 1995 年第一次发布内测版本，到目前最新的 8.0 版本，已有 29 年历史。在 MySQL 的发展历史中，需要关注的有 3 件事。

- 2000 年，MySQL 公开了源代码，成为开源软件，随后成为网站后台数据库的首选。从那时起，它一直占据网站数据库的大部分份额。
- 2008 年，Sun 公司收购了 MySQL，2010 年 Oracle 公司又收购了 Sun 公司。MySQL 的创始人蒙蒂·维德纽斯（Monty Widenius）不满 Oracle 公司的政策，离开了 Oracle 公司，并创立了一个新的开源数据库产品 MariaDB。这个产品与 MySQL 兼容，有许多使用 MySQL 的用户转而使用 MariaDB。
- 2011 年，Oracle 公司为避免 MySQL 与自家的 Oracle 数据库产品冲突，将 MySQL 分为企业版和社区版两种产品，企业版是收费的，而社区版是免费的。

1.2.1 版本介绍

本书采用 MySQL 8.0.36 版（2024-01-16 发布）。从 MySQL 5.5 开始，MySQL 的基本功能已经十分完善，MySQL 3 到 MySQL 8 之间各版本的简介如表 1-3 所示。

表 1-3 MySQL 3 到 MySQL 8 之间各版本的简介

版本	发布年份	新增功能
3.23	2001	正式进入大众视野，成为一种流行的数据库管理系统
4.0	2003	引入了 InnoDB 存储引擎，支持事务，但还不是默认的存储引擎
5.0	2006	增加存储过程、游标、触发器、查询优化以及分布式事务功能等
5.1	2008	增加事件（一种定时任务）、分区，基于行的复制等
5.5	2010	一次重要的升级，默认存储引擎更改为 InnoDB、提高性能和可扩展性
5.6	2012	InnoDB 性能加强，以及其他改进
5.7	2015	提升 MySQL 安全性，以及其他改进
8.0	2016	首发于 2016。2024-01-16 发布 8.0.36 版，这是最新的稳定版本（长期支持版）
8.3	2024	创新版，用于尝试一些最新的功能。8.3 版与 8.0.36 版同时于 2024-1-16 发布

注：版本 6 和 7 是内部版本，没有正式发行。

1.2.2 软件下载

从 MySQL 的主页（https://www.mysql.com/）下载 MySQL 软件，MySQL 支持 Windows、Linux、Mac OS 等多种操作系统，本书选择基于 Windows 的 MySQL，文件名是 mysql-installer-community-8.0.36.0.msi，文件大小是 285MB。也可以从本书主页的网盘链接下载（http://ngweb.org/mysql/?p=8）这个文件。

该安装文件包含了一组软件，本书安装的是其中两个软件，MySQL Server 和 MySQL Workbench，前者是 MySQL 的核心，是必选的，后者是一个客户端工具，是可选的。如果不安装 MySQL Workbench，则需要安装其他的客户端工具，常用 MySQL 客户端工具如表 1-4 所示。

表 1-4 常用 MySQL 客户端工具

客户端名称	说明
MySQL Workbench	免费，MySQL 官方提供，仅支持 MySQL，功能强大，支持数据库开发全过程，包括数据建模、服务器配置和监视、安全管理、备份和恢复自动化、审计数据检查以及向导驱动的数据库迁移
MySQL 命令行	MySQL Server 内置，无需安装，不是图形界面，将在单元 8 “8.1 MySQL 命令行” 中讲解
Navicat	商业版，PremiumSoft 公司提供，还支持 Oracle、SQL Server 和 SQLite 等，功能丰富，使用方便
dbForge	商业版，devart 公司提供，还支持 Oracle 和 SQL Server 等，功能丰富，使用方便
phpMyAdmin	免费，由社区开发和维护，仅支持 MySQL，是一种网页版，需要 PHP 运行环境，功能较简单

1.2.3 安装和配置 MySQL

双击下载的安装文件 mysql-installer-community-8.0.36.0.msi 开始安装，如图 1-2 所示，从图中左侧列表可以看到，安装过程分为五个步骤。

1. Choosing a Setup Type 选择安装类型

第一步是选择安装类型，因为要同时安装 MySQL Server 和 MySQLWorkbench，所以选择最后一项“Custom”（自定义安装），如图 1-2 所示，然后单击 Next 按钮。

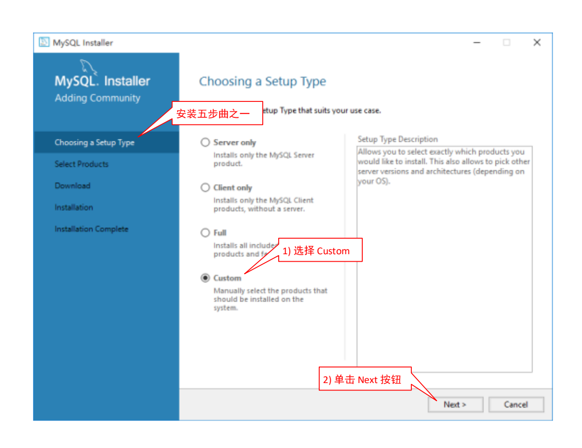


图 1-2 安装步骤之一：选择安装类型

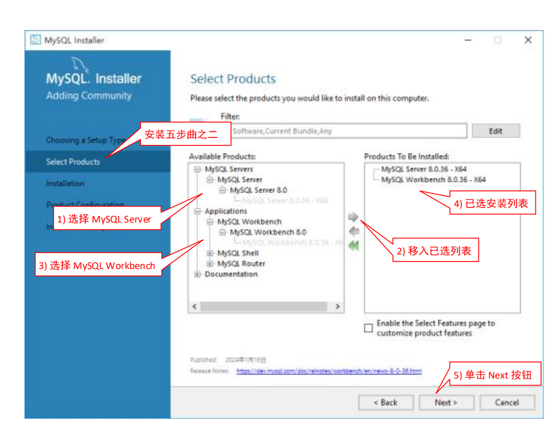


图 1-3 安装步骤之二：选择产品

2. Select Products 选择产品

接下来是选择要安装的产品，在左侧的产品列表中，多次单击“+”号，找到要安装的产品 MySQL Server 8.0.36 X64，再单击右箭头，将其移入右侧的已选列表。找到 MySQL Workbench 8.0.36 X64，将其移入已选列表，这时已选安装列表应该有两项，如图 1-3 所示，然后单击 Next 按钮。

3. Installation 安装

现在开始安装，如图 1-4 所示，单击 Execute 按钮，安装过程大约需要 2-3 分钟，并有进度提示。安装完成后，单击 Next 按钮进入下一个界面。

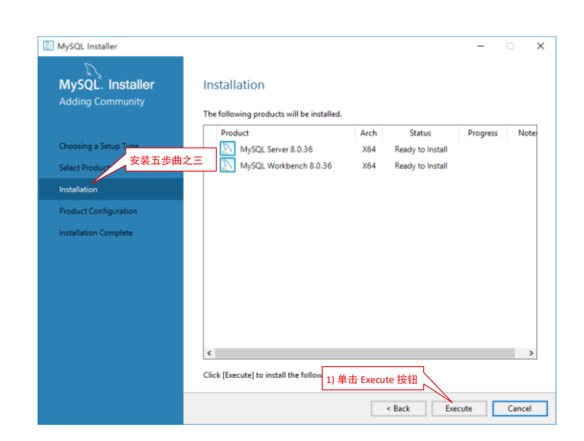


图 1-4 安装步骤之三：安装

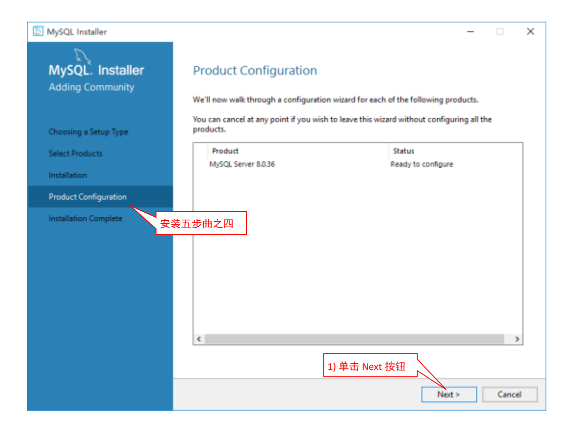


图 1-5 安装步骤之四：产品配置

4. Product Configuration 产品配置

安装完成后将要进入产品配置，如图 1-5 所示，产品配置是另外一组界面，配置的选项有很多，但是在本书里全部采用默认配置（除了设置系统管理员的密码），配置完成后再回到产品配置这一步。

单击 Next 按钮，进入产品配置界面，如图 1-6 所示，从图中左侧列表可以看到，产品配置过程分为

六个步骤。

1) Type and Networking 服务器类型和网络

这一步是配置服务器类型和网络，服务器类型有开发用，或者是生产用等。网络配置需要保留 TCP/IP 的勾选状态，端口号保留默认的 3306。采用默认值，无需修改，如图 1-6 所示，然后单击 Next 按钮。

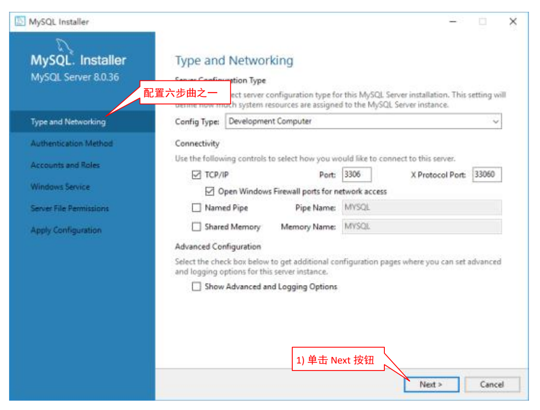


图 1-6 配置步骤之一：服务器类型和网络

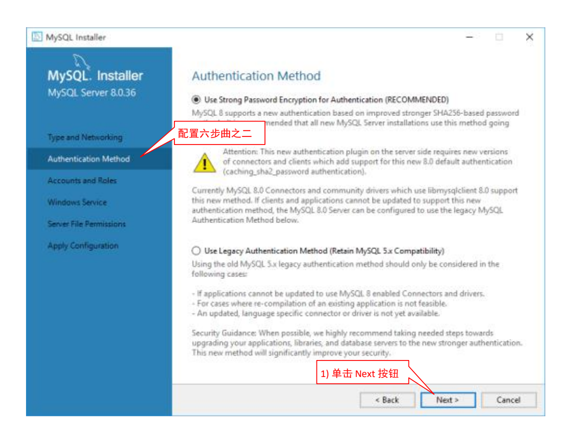


图 1-7 配置步骤之二：身份认证方法

2) Authentication Method 身份认证方法

身份认证方法有两种，一种是兼容 MySQL 5.x 的老的身份认证方法，一种是新的强密码加密方法。建议使用新的身份认证方法。采用默认值，无需修改，如图 1-7 所示，然后单击 Next 按钮。

3) Accounts and Roles 账号和角色

这一步是设置系统管理员账号的密码，系统管理员是整个数据库中权限最高的账号，所有其他用户的权限都要由它来授予。

系统管理员账号的名称是 root，密码在这一步设置，对于初学者，建议将密码统一设置为 sasa，以免忘记，这是系统管理员（Ssystem Administator）的首字母，重复两次是因为密码必需至少 4 位长，如图 1-8 所示，然后单击 Next 按钮。

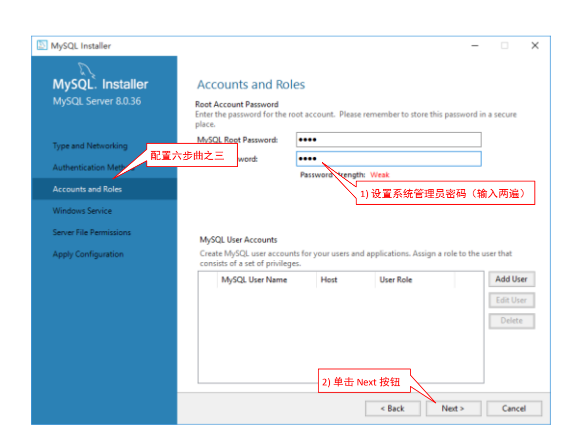


图 1-8 配置步骤之三：账号和角色

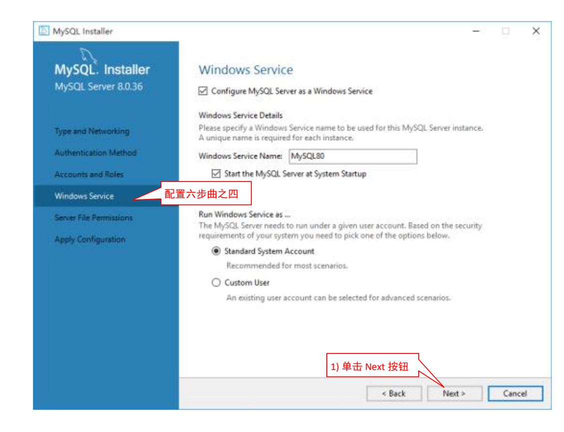


图 1-9 配置步骤之四：Windows 服务

4) Windows Service 服务

这一步设置服务的名称，默认是 MySQL80，并设置为系统开机启动时自动启动。采用默认值，无需修改，如图 1-9 所示，然后单击 Next 按钮。

5) Server File Permissions 服务器文件许可

这一步是设置服务器上数据库文件的访问许可，采用默认值，无需修改，如图 1-10 所示，然后单击

Next 按钮。

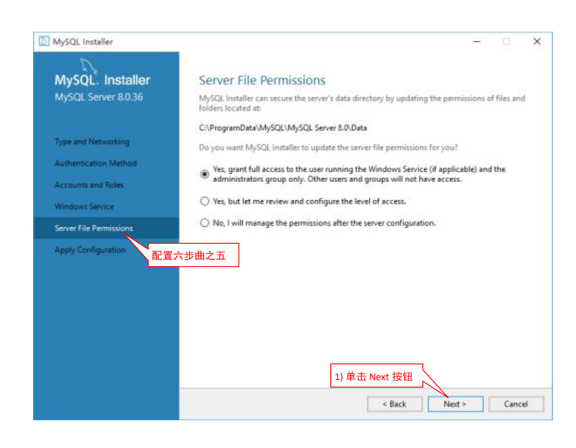


图 1-10 配置步骤之五：服务器文件许可

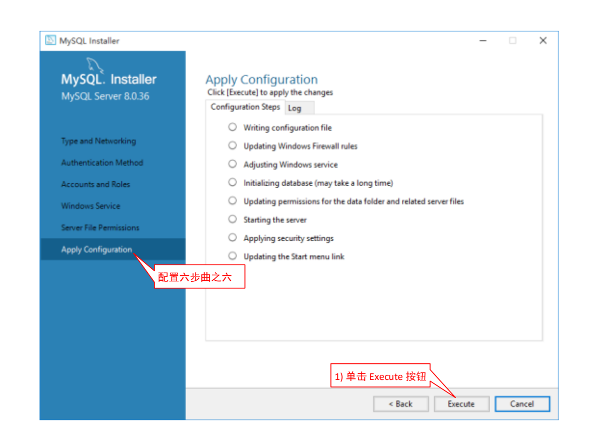


图 1-11 配置步骤之六：应用配置

6) Apply Configuration 应用配置

配置完成后，需要应用所作的配置，如图 1-11 所示，单击 Execute 按钮。配置过程大约需要 2-3 分钟，并有进度提示。配置完成后，单击 Finish 按钮关闭配置窗口，结束配置过程。

5. 安装完成

配置结束后，回到 MySQL 安装界面的第四步，单击 Next 按钮，进入安装界面的最后一步，如图 1-12 所示，直接单击 Finish 按钮，安装全部完成。

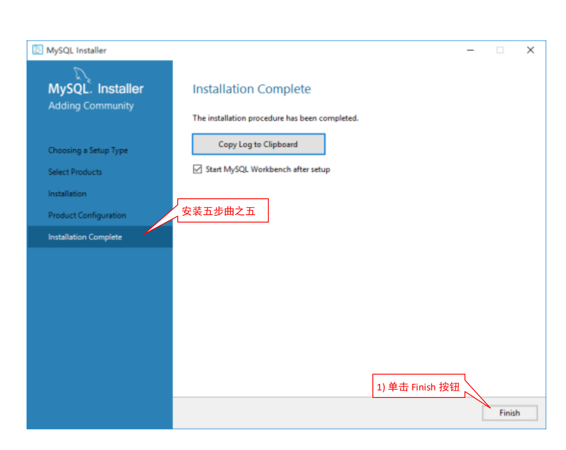


图 1-12 安装步骤之五：安装完成

至此，MySQL Server 和 MySQL Workbench 全部安装完成。



安装过程出现问题时，请访问附录 E 的在线资源网站，在问题解答中会有各种问题的解答。如果您的问题没有找到，可通过您的老师将问题反馈给作者。

1.3 MySQL 的使用

1.3.1 MySQL 客户端

上一节安装了 MySQL Server 和 MySQL Workbench，MySQL Server 是 MySQL 的核心，但是不能直接使用 MySQL Server，必须通过一个管理工具来使用 MySQL Server，这个管理工具也称为 MySQL 客户端，本书使用 MySQL Workbench 客户端。

1.3.2 MySQL 的界面

1. 开始菜单中的 MySQL

安装完成后，在“开始菜单”中只能看到安装好的 MySQL Command Line、MySQL Installer 和 MySQL Workbench，而看不到 MySQL Server，如图 1-13 所示。在 MySQL 菜单组内有下述四项。

- MySQL 8.0 Coomand Line Client：MySQL 官方的命令行管理工具，这是内置的管理工具，一般由熟练的数据库操作人员使用，在单元 8“8.1 MySQL 命令行”中讲解。
- MySQL 8.0 Coomand Line - Unicode：与上一项相同，增加了对中文的支持。
- MySQL Installer - Community：这是 MySQL 的安装程序，用于卸载 MySQL 或进行重新配置，修改配置参数。配置过程在前述的“配置 MySQL”中已经作过讲解。
- MySQL Workbench 8.0 CE：MySQL 官方的图形界面管理工具，本书使用这个管理工具对 MySQL 进行操作和管理。

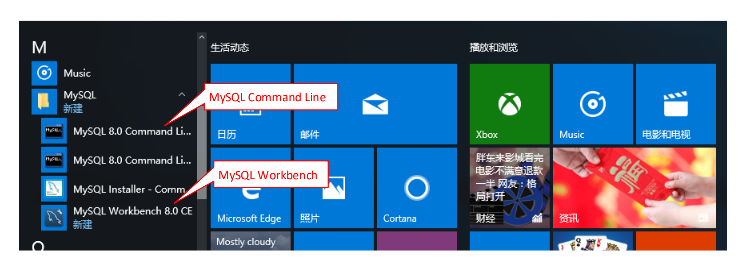


图 1-13 安装 MySQL Server 和 MySQL Workbench 后的“开始菜单”

2. MySQL Server 的界面

MySQL Server 是没有界面的，图 1-13 所示的开始菜单中只有 MySQL 的管理工具和安装程序的入口，而没有 MySQL Server 的入口，这是因为 MySQL Server 是在后台持续运行的，它随计算机开机而启动，随计算机关机而停止工作。作为 MySQL 数据库的使用者和管理者，通常无需关注 MySQL Server 本身的运行状态，除非它出现了故障。

3. MySQL Workbench 的界面

本书主要讲解通过 MySQL Workbench 来使用和管理 MySQL Server。

1) 主界面

打开 MySQL Workbench，它的主界面如图 1-14 所示，其中有 MySQL 工作台、MySQL 建模工具和 MySQL 迁移工具。

- MySQL 工作台：用于使用和管理 MySQL Server，全书使用它进行讲解和操作。
- MySQL 建模工具：用于数据建模，在单元 3 的“3.5 建模工具的使用”一节讲解。
- MySQL 迁移工具：用于与其他数据库之间的迁移，例如将 Oracle 或 SLQ Server 数据库迁移到 MySQL 数据库，或将 MySQL 数据库迁移到 Oracle 或 SLQ Server 数据库。

2) 连接（登录）MySQL 服务器

在使用 MySQL 之前，需要连接登录到 MySQL Server，这就是 MySQL Connection。连接时需要提供账号和密码，因此连接的过程就是一个登录的过程。

图 1-14 所示的左下方列出所有 MySQL Connections，目前只有一个默认的本地连接，称为 Local instance MySQL80，Local instance 表示这个 MySQL 是安装在本地计算机上的一个实例，MySQL80 是服务的名称，是在如图 1-9 所示的界面上配置的。

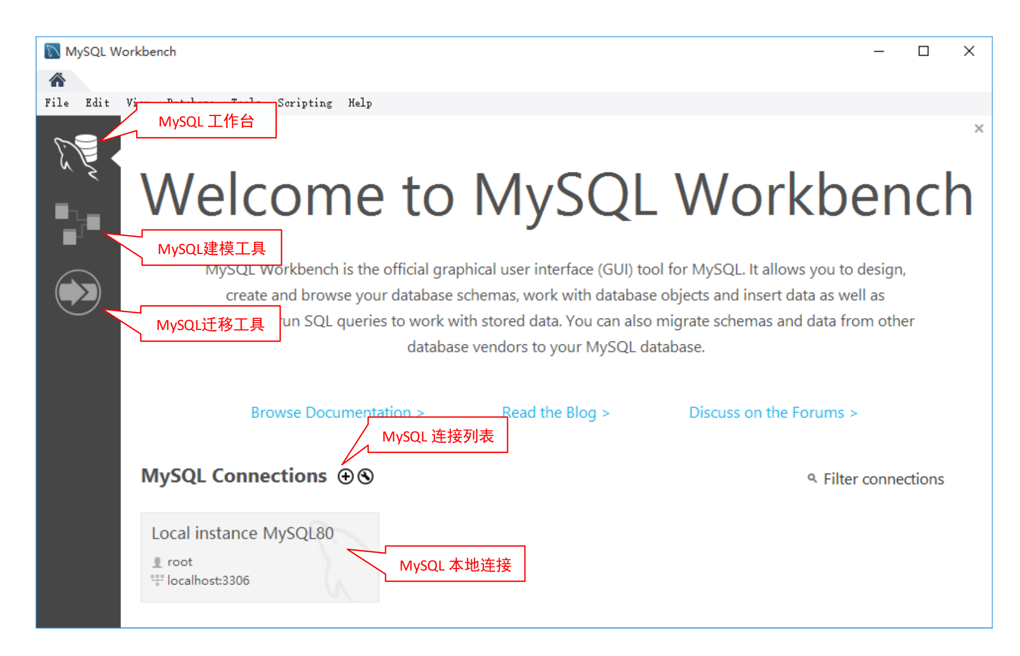


图 1-14 MySQL Workbench 启动界面

单击 Local instance MySQL80，将弹出一个对话框，如图 1-15 所示。连接的账号是 root，也就是系统管理员，因此这时要输入 root 账号的密码，这个密码是在如图 1-8 所示的界面上配置的。填入 root 系统管理员的密码，可能是图 1-8 建议的 sasa，也可能是读者自己定的密码，再勾选 Save password in vault 多选框，如果不勾选，则下次连接到 MySQL 服务时还要填写密码。

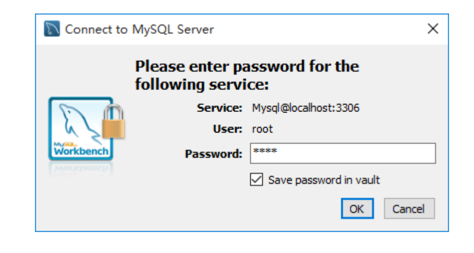


图 1-15 连接登录填写 root 账号的密码

连接登录成功后，进入 MySQL Workbench 界面，首先看到的是全局管理界面（Administration），如图 1-16 所示，这是用于全局管理的界面。

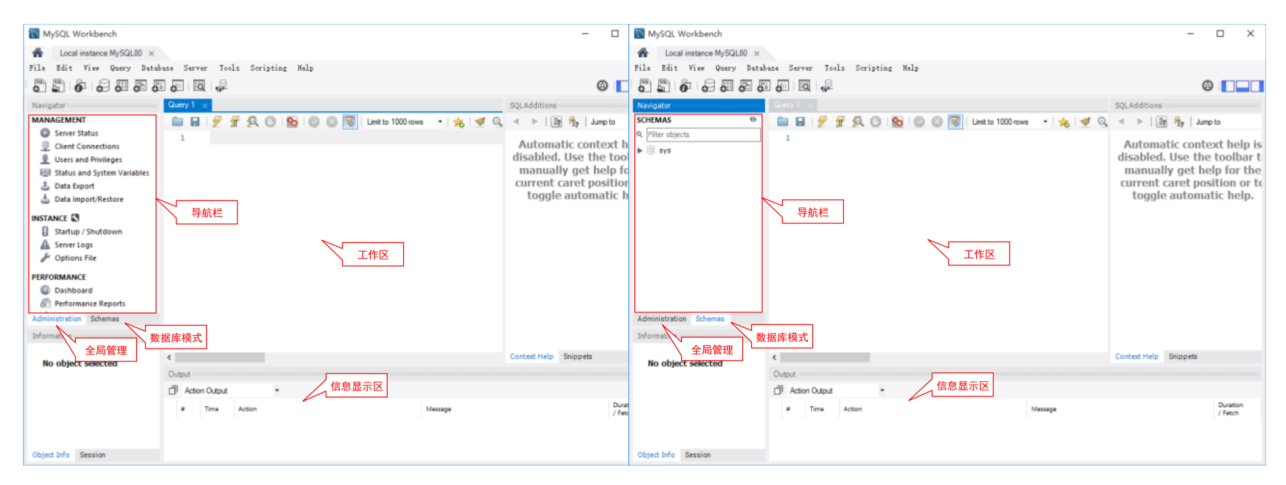


图 1-16 全局管理（Administration）界面　　图 1-17 数据库模式（Schemas）界面

3) 全局管理界面

全局管理界面的导航栏有如下三组管理菜单，如图 1-16 所示。

- MANAGEMENT 管理：这是具体的管理，主要有服务器的状态，连接的管理，用户账号和授权，数据的导出（备份）和导入（恢复）等。
- INSTANCE 实例：服务的启动和停止，服务日志记录等。

- PERFORMANCE 性能：仪表盘，性能报告等。

4) 数据库模式界面

MySQL Workbench 的另外一个主界面是数据库模式（Schemas）界面，如图 1-17 所示，这是本书使用 MySQL 的主要界面，90%以上的篇幅都是同 Schemas 界面有关的。

数据库模式界面的导航栏是 Schemas 的列表，这个列表就是数据库列表，目前显示的只有 sys 数据库，这是一个内置数据库，包含了 MySQL 的内部运行数据，管理员可以通过它来观察 MySQL 的运行情况，并进行性能评估和优化等。

这里要解释一下数据库模式（Schema）和数据库（Database）这两个术语的区别，它们是两个有些不同但又密切相关的术语，数据库模式（Schema）主要是指数据库的概念，它包括数据结构，但不包括具体的数据，是一种比较抽象的描述。数据库（Database）是更加具体的概念，数据库包括了数据结构和具体的数据，是能够具体操作的数据库对象的集合。



数据库对象包括表（Table）、视图（View）、存储函数（Function）、存储过程（Stored Procedure）和触发器（Trigger）等，都是本书讲解的内容。

在 MySQL 数据库管理系统中，可以认为 Schema 和 Database 是两个相同的术语，可以互相替代。如图 1-17 所示的 sys 数据库也可以称为 sys 模式，通常还是称为 sys 数据库，而不称为 sys 模式，因为 sys 模式这个说法只在 MySQL Workbench 中使用。如果在 Oracle 或 SQL Server 数据库管理系统中，Schema 和 Database 有不同的使用场景，不能混用。

1.3.3 使用 MySQL

作为一个入门练习，下面学习使用 MySQL 的过程。在这个“Hello，数据库”项目里，把如图 1-18 所示的数据保存到数据库中，这些数据原来是保存在 Excel 文件中的。



	A	B	C	D
1	作者	书名	价格	出版社
2	黄能耿	MySQL数据库应用实战教程	59.8	人民邮电出版社
3	周德伟	MySQL数据库基础实例教程	49.8	人民邮电出版社

图 1-18 “Hello，数据库”的数据

1. 分析数据

开发任何一个项目，第一步是分析数据，也称为需求分析。根据数据库以及表的含义，将数据库命名为 book，将表命名为 book_list，然后还要对数据进行分析和设计。这些数据共有 4 列，为了标识每一行，需要增加一个标识列，这个列称为主键，通常用 ID（identity 的前两个字母，含义是标识）表示。

对每一列数据进行分析，确定数据的类型和要求，结果如表 1-5 所示。其中 int 表示整型，varchar 表示可变长字符串，括号中的数字是最大长度，长度 45 是默认值。decimal 表示实数，括号中的数字分别是有效位数和小数位数。

表 1-5 “Hello，数据库”中 book_list 表每一列数据的要求

列标题	要求	数据类型	建议的列名
主键（ID）	整数，不允许重复，不能为空，自动编号	int	id
作者	可变长字符串，最多 45 个字符	varchar(45)	author
书名	可变长字符串，最多 100 个字符	varchar(100)	title
价格	实数，两位小数	decimal(9,2)	price
出版社	可变长字符串，最多 45 个字符	varchar(45)	publisher



主键（ID）是额外添加的列。主键（PK，Primary Key）不能为空（NN，Not Null），通常是整型，可以是自增的（AI，Auto Increment），在任何一张表中，都必须添加，无一例外。



下面根据表 1-5 的要求，将图 1-18 的数据保存到数据库中。读者可以在实训 1-1 的指导下完成下面的步骤（见附录 C）。

2. 创建数据库

首先创建一个名为 book 的数据库，如图 1-19 所示。

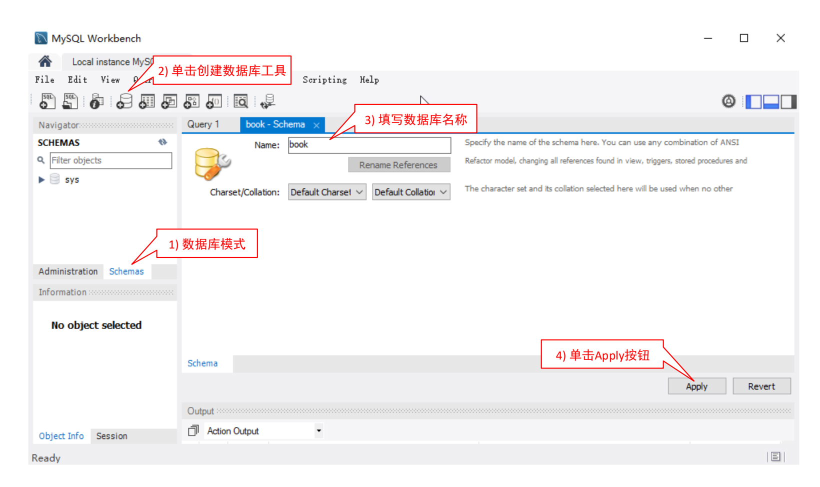


图 1-19 创建 book 数据库

具体操作步骤如下。

1) 打开 MySQL Workbench，连接到 MySQL 后，进入“数据库模式”界面。
2) 单击“创建数据库”工具按钮，这时打开创建数据库选项卡。
3) 在选项卡的 Name 处，填写数据库的名称 book。
4) 单击“Apply”按钮，这时弹出一个对话框，显示刚才的操作所生成的一条“创建 book 数据库”的语句“CREATE SCHEMA `book` ;”，如图 1-20 所示。
5) 在如图 1-20 所示的对话框上单击“Apply”按钮，确认执行这条语句，创建 book 数据库。然后单击 Finish 按钮，关闭对话框。

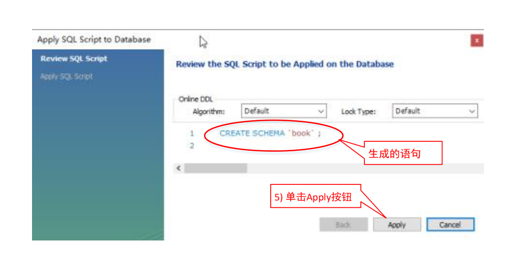


图 1-20 确认执行“创建 book 数据库”语句

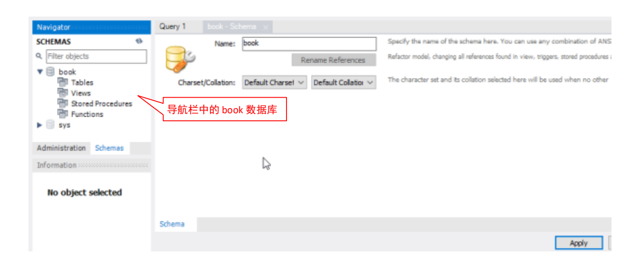


图 1-21 新创建的 book 数据库

至此，创建 book 数据库完成，在左侧的导航栏上列出了新创建的 book 数据库，单击其前面的小三角符号，可以看到 book 数据库还包含了 Tables 等数据库对象，如图 1-21 所示。

3. 创建表

这一步是在 book 数据库中创建名为 book_list 的表，如图 1-22 所示。

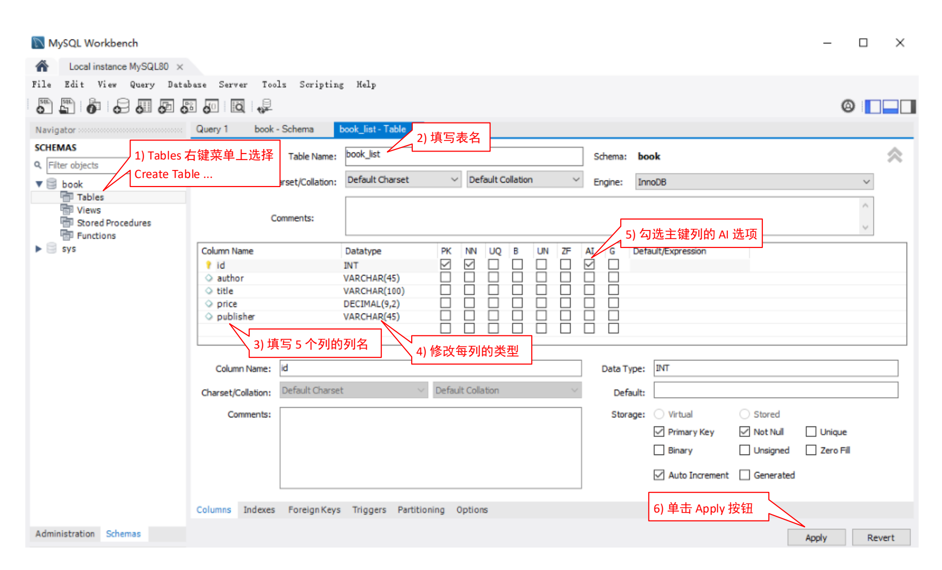
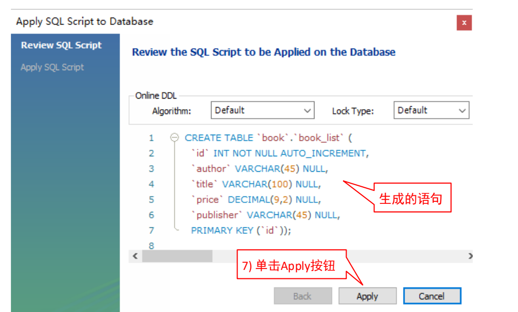


图 1-22 创建 book_list 表

具体操作步骤如下。

1) 在 book 数据库的 Tables 项下，通过右键菜单的 Create Table…，打开创建表的选项卡。
2) 在选项卡的 Table name 处填写表的名字 book_list。
3) 在中部的 Column name 处，逐行添加并填写列名，注意当单击 Column name 下方的空白处时，会自动添加一行空行，并预置了默认的列名，修改这个列名为表 1-5 建议的列名。
4) 根据表 1-5 对数据类型的要求，修改每一列的数据类型。
5) 勾选主键列（列名是 id）的 AI 选项，这是自动增量（Auto Increment）的缩写，其他两个选项已经默认勾选了，它们是 PK 和 NN，分别是主键（Primary Key）和非空（Not Null）的缩写。
6) 核对上述输入的参数后，单击 Apply 按钮，这时弹出一个对话框，显示刚才操作所生成的一条“创建 book_list 表”的语句“CREATE TABLE `book`.`book_list`……”，如图 1-23 所示。
7) 在如图 1-23 所示的对话框上单击“Apply”按钮，确认执行这条语句，创建 book_list 表。然后单击 Finish 按钮，关闭对话框。



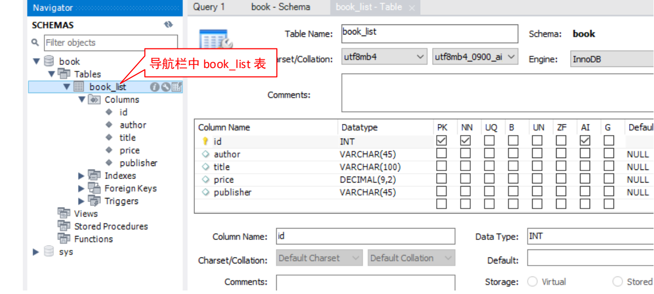


图 1-23 确执行“创建 book_list 表”语句　　　　图 1-24 新创建的 book_list 表

至此，创建 book_list 表完成，在左侧的导航栏上列出了新创建的 book_list 表，单击其前面的小三角形符号，可以看到表的各个组成部分，包括各个列的列名等详细信息，如图 1-24 所示。

4. 输入数据

接下来向 book_list 表输入数据，如图 1-25 所示。

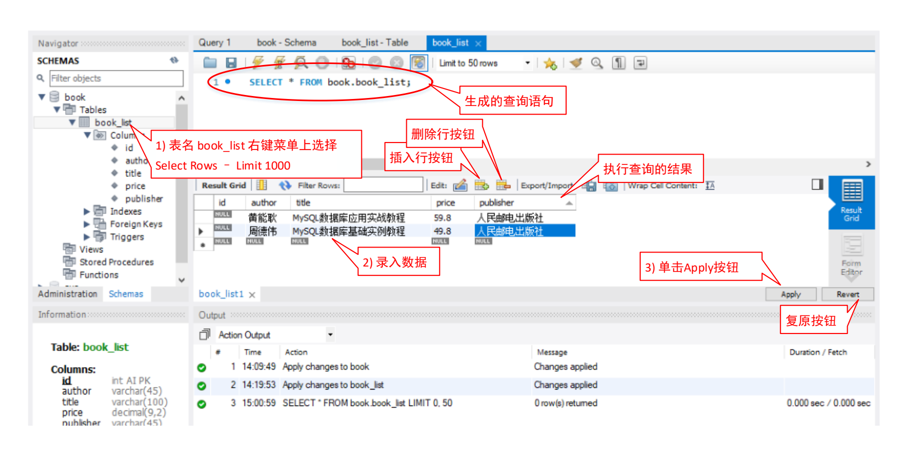


图 1-25 输入数据

具体操作步骤如下。

1) 在表名 book_list 的右键菜单中选择 Select Rows - Limit 1000，打开一个新的选项卡，这个选项卡的上方是一条生成的语句，下方是执行这条语句的结果，目前执行结果为空，因为表中还没有数据。

2) 在下方执行结果的区域内输入图 1-18 所示的测试数据，注意两点，一是单击底色为灰色的 null（这个符号表示没有数据）就能直接输入数据或直接添加新行，二是主键列（id）不要输入任何数据，因为它是自动生成的。

3) 数据输入完成，并确认无误后，单击 Apply 按钮，这时弹出一个对话框，显示刚才的操作所生成的多条“输入数据”语句“INSERT INTO `book`.`book_list` ……”，每行数据（即每本书）对应一条语句，如图 1-26 所示。

4) 在如图 1-26 所示的对话框上单击“Apply”按钮，确认执行这条语句，保存这些数据。然后单击 Finish 按钮，关闭对话框。



图 1-26 确认执行“输入数据”的语句

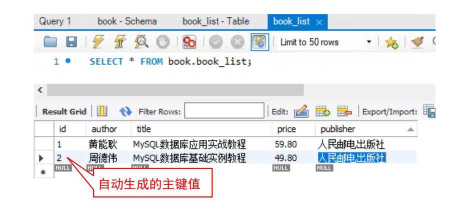


图 1-27 输入完成的数据

至此，输入数据完成，刚才输入数据的区域成为了输入完成的结果，在结果中显示出自动生成的主键（id）值，如图 1-27 所示。

因为主键（id）值是自动生成的，因此保证了这个值是递增的，并且不会重复，根据这个值 ，就能唯一标识一本书的信息，不可能出现两本书拥有相同主键值的情况。

输入数据时注意以下几点。

1. 主键是自动赋值的，不需要输入，也不应该输入，以防输入了重复的主键值。
2. 输入的数据应该与列的数据类型一致，例如价格只能输入数字，而不能输入字母。
3. 可以通过“插入行”或“删除行”按钮（参见图 1-25 中部）来插入行或删除行。
4. 如果想放弃所作的修改（包括插入新行或删除行），可以单击 Revert 按钮复原（撤回）。

5. 查询数据

前面的步骤完成了数据库和表的创建，数据也输入并保存好了，下面的任务是使用这些数据，以体现

数据的价值，如图 1-28 所示。

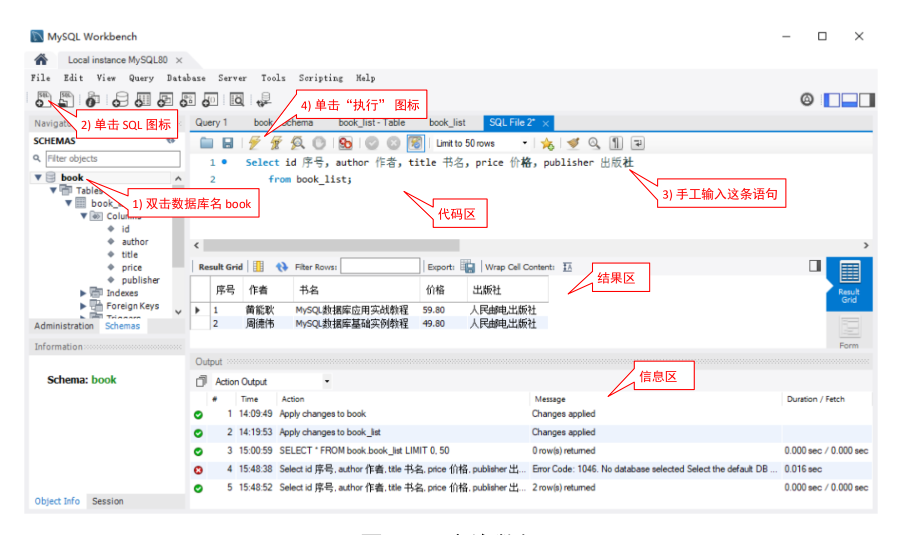


图 1-28 查询数据

具体操作步骤如下。

1) 双击数据库名 book，使其名字加粗显示，成为当前打开的数据库。这个步骤保证了后续操作是在 book 数据库内执行的，而不是在其他数据库内执行。

2) 单击工具栏的第一个图标“SQL 图标”，打开“SQL File”选项卡。

3) 在代码区手工输入一条查询数据的语句，如下所示。

```
Select id 序号, author 作者, title 书名, price 价格, publisher 出版社
    from book_list;
```

4) 单击“执行”图标，这个图标像一个闪电，表示一种执行力。

这时，在结果区显示查询语句的执行结果，在信息区显示相关信息，如图 1-28 所示。

有错误时会在信息区显示出错信息。例如图 1-28 所示的信息区第 4 行是一条出错信息“Error Code: 1046. No database selected”，意思是“没有选择数据库”，这是因为忘记双击数据库名 book 而引起的。

在查询结果中，将主键的标题显示为序号，实际上主键和序号是有一定区别的，主键是有严格数学定义的，它必须是唯一的，并且不允许为空，但不一定是连续的，中间可以跳过一些编号。而序号是日常使用的，应该是唯一的，不为空的，但是通常是连续的，不能跳过一些编号。

6. 在浏览器查看项目结果



在数据库的实际应用中，要通过数据库应用程序来访问数据库。根据附录 D 的说明，打开名为“项目 1a Hello，数据库项目”的数据库应用程序，如图 1-29 所示。



图 1-29 “Hello，数据库”项目的应用程序界面

在浏览器中，可以查看在 MySQL Workbench 输入的数据。在浏览器中添加或修改的数据，也能够在 MySQL Workbench 中看到。



注意：如果读者创建的表结构与表 1-5 的要求不同，例如表名不一致，或列名不一致，应用程序就会找不到这张表或这个列，导致应用程序无法正常运行，这时可以从应用程序的“运行日志”中查看错误信息，找出错误原因，需要时还要修改表结构。

1.4 MySQL 服务器与 MySQL 客户端

上述过程就是程序员通过 MySQL 客户端与 MySQL 服务器实现交互的过程。在上一节安装的 MySQL Server 是 MySQL 服务器，MySQL Workbench 是 MySQL 客户端，程序员通过 MySQL 客户端来管理和操纵 MySQL 服务器上的数据库。

程序员在 MySQL 客户端进行操作，如图 1-30 所示，客户端将这些操作翻译为一条或多条 SQL 语句，第一次按 Apply 按钮时，MySQL 客户端显示这些语句，第二次按 Apply 按钮时就是执行这些 SQL 语句。MySQL 客户端将 SQL 语句发送给 MySQL 服务器，MySQL 服务器执行 SQL 语句，根据 SQL 语句的要求对数据库进行操作，并返回有关执行的信息和执行的结果，然后 MySQL 客户端显示这些信息。

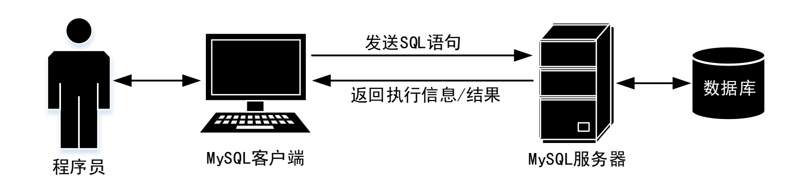


图 1-30 MySQL 客户端与 MySQL 服务器的交互过程

程序员继续操作并执行 SQL 语句，这样一问一答，就实现了对 MySQL 数据库的操作和管理。

MySQL 客户端只是一个界面，真正执行 SQL 语句的是 MySQL 服务器。服务器根据 SQL 语句的要求，将数据保存到数据库中，或从数据库中查询数据。因此，MySQL 的核心是 MySQL 服务器，用户可以用不同的 MySQL 客户端连接到 MySQL 服务器，向 MySQL 服务器发送 SQL 语句，并接收 SQL 语句执行的结果。

1.5 理解数据库

本节讲解与数据库有关的基本概念，包括数据库（Database，DB）、数据库管理系统（Database Management System，DBMS）和数据库系统（Database System，DBS）等概念，以及 SQL（Structured Query Language 和 NoSQL（Not Only SQL）等术语。

1.5.1 数据库

首先回顾一下“Hello，数据库”都做了些什么，在这个例子中，创建了一个名为“book”的数据库，在这个数据库里创建了一张名为“book_list”的数据表（简称表），向表中插入了两行数据。并通过一条查询语句将数据以合适的方式展现出来，例如列的名称在表中是 author、title 和 price 等，但是在查询时可以根据需要，显示为作者、书名和价格，也可以显示为其他标题，这些数据还可以通过网络（例如浏览器）而被其他人访问。

从这个例子可以总结出数据库的以下三个特征。

- 有组织的数据：在创建表时定义了数据的组织结构，即结构化的数据。
- 数据的集合：可以设计任意多张表，每张表可以输入任意多行数据。

- 可共享的数据：数据可以通过网络被共享。

因此，数据库是存储在计算机上的有组织的、可共享的数据的集合。

几乎所有网站、手机 App 以及多数应用软件的后台都有一个数据库，这些数据库保存了用户登录的账号和密码，保存了需要动态显示的内容。例如一个银行 App 中显示的存取款记录全部来自数据库。

1.5.2 数据库管理系统

数据库是数据的集合，对数据来说，不同的组织形式和不同的处理方式将会对操作的效率和处理的结果产生不同的影响。因此，用户需要借助一个软件工具来对数据进行组织和处理，这个软件工具就是数据库管理系统。

数据库管理系统（Database Management System，DBMS）是为管理数据库而设计的通用软件系统，例如本书采用的 MySQL，就是一个数据库管理系统。数据库管理系统应该具有如下功能，对这些功能的讲解构成了本书的主要内容。

1. 数据定义功能

定义数据库中数据的组织形式，即数据结构（Data Structure），如定义数据库、表、视图和索引等。DBMS 提供了数据定义语言（Data Definition Language，DDL）来实现这个功能，例如在“Hello，数据库”中创建数据库、创建表、指定列名、列的数据类型，以及指定表的主键等。

2. 数据操纵功能

操纵数据库中的数据，实现对数据库中数据的插入、更新与删除等操作。DBMS 提供了数据操纵语言（Data Manipulation Language，DML）来实现这个功能，例如在“Hello，数据库”项目中向 book_list 表输入数据、修改或删除数据。

3. 数据查询功能

查询数据库中的数据，实现查找、统计和分析等各种灵活的查询操作。DBMS 提供了数据查询语言（Data Query Language，DQL）来实现这个功能，例如在“Hello，数据库”中，图 1-28 所示的数据就是通过查询而得到的。

4. 数据控制功能

确保数据库的安全性、完整性，以及数据的备份、恢复，提供完善的运行管理机制，确保数据库的稳定运行。DBMS 提供了数据控制语言（Data Control Language，DCL）来实现这个功能。

5. 本书与数据库管理系统

对上述四大类功能的讲解构成了本书的主要内容，单元 3 详细讲解“数据定义”，单元 4 详细讲解“数据操纵”，单元 5 和单元 6 详细讲解“数据查询”，单元 8 讲解“数据控制”，在讲解数据定义之前，先要讲解关系数据库的一些基础知识（单元 2），而在讲解数据查询之后，还要讲解数据库编程（单元 7），通过编程技术把数据定义、数据操纵和数据查询结合起来，本书的最后一个单元是“项目实战”，通过几个实战项目，演示了综合运用这些技术，开发一个功能完整的应用系统的过程。

1.5.3 数据库系统

数据库系统（Database System，DBS）由计算机软硬件系统、数据库管理系统、数据库、数据库应用程序、使用人员 5 个部分组成，如图 1-31 所示。

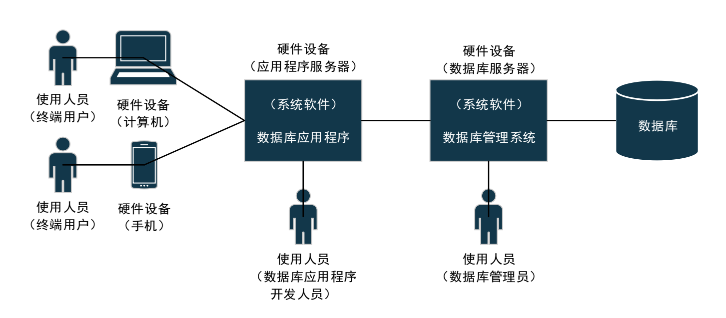


图 1-31 数据库系统的组成

1. 计算机软硬件系统

计算机硬件系统是指计算机设备、服务器（数据库服务器、应用程序服务器）、网络设备等，计算机软件系统是指操作系统和软件支撑环境。图 1-31 中所用到的硬件、网络、系统软件都属于这一类。

2. 数据库管理系统

数据库管理系统是管理和操纵数据库的软件系统，是数据库系统的核心，上一小节对此做过详细的讨论，数据库管理系统（软件）是安装在数据库服务器（硬件）上的。

3. 数据库

实际的数据库是由数据库管理系统创建的，包含了数据结构和数据，还有视图、索引、存储函数和存储过程等等。在内部，它保存在一组文件中。

数据库管理系统是通用软件，可用于各种应用需求。而数据库则是针对具体的应用需求，由开发人员采用某种数据库管理系统设计的满足应用需求的数据结构，以及保存在其中的数据，可供进一步的处理和应用。

在 MySQL 中可以创建多个数据库，通常每个数据库是一个项目，每个项目会有不同的用途，例如学生管理数据库、财务管理数据库、图书借阅数据库等。



通常情况下，数据库一词可以用来表示数据库这个概念、实际的数据库、数据库管理系统、数据库系统以及数据库技术等多种含义，需要根据应用场景进行判断。

4. 数据库应用程序

数据库应用程序是为方便用户操纵和维护数据库中的数据而开发的应用程序，提供友好的界面，允许用户方便地插入、更新、删除数据，以及查询数据库中的数据，数据库应用程序通过数据库管理系统对数据库中的数据进行操作。数据库应用程序是安装在应用程序服务器上的。

在开发“Hello，数据库”项目时，已经体验过这个项目的应用程序，如图 1-29 所示。常用的数据库应用程序开发语言和技术有 HTML5、JavaScript、Java、PHP、Delphi 和 C#等。本书的应用程序是用 Java 语言和 HTML5 + JavaScript + CSS 技术开发的，在单元 9 中将作详细讲解，但是读者不需要任何 Java 或 HTML5 的知识，只需要数据库的知识，编写 SQL 语句，就能开发出一个功能完整的项目。

5. 使用人员

使用数据库的人员分为 3 类：数据库管理员、数据库应用程序开发人员和终端用户。

- 数据库管理员（Database Administrator，DBA）是管理数据库系统的人员，他们的主要任务是负责数据库的日常维护和安全，保障数据库的正常运行。
- 数据库应用程序开发人员根据数据库应用的具体需求，设计数据库的数据结构，设计和编写数据库应用程序中各功能模块的界面与程序代码。

- 终端用户是最终使用数据库应用程序的人员，终端用户可以通过计算机或手机来使用数据库应用程序。例如，对于医院管理系统，终端用户是医生和护士等；对于学生管理系统，终端用户是教师和学生，以及管理人员。

学习 MySQL 数据库的本书读者是作为数据库管理员或者是数据库应用程序开发人员的身份出现在数据库系统中的。

1.5.4 SQL 和 NoSQL

数据库管理系统分为两大类：关系数据库管理系统（简称 SQL 数据库）和非关系数据库管理系统（简称 NoSQL 数据库）。

1. 结构化、半结构化和非结构化数据

要理解关系数据库管理系统和非关系数据库管理系统，就要先了解结构化、半结构化和非结构化数据的概念。

1) 结构化数据

结构化数据是指严格遵循数据模型、易于搜索和组织的数据，通常存储在关系数据库中。结构化数据的例子有 Excel 电子表格中的数据、SQL 数据库中的数据。

2) 半结构化数据

半结构化数据不遵循严格的数据模型，但仍具有一定的组织结构，使其便于处理。半结构化数据的例子有 JSON 和 XML 文件、HTML 文档、以及日志文件。

3) 非结构化数据

非结构化数据没有预定义的数据模型，通常是文本或多媒体内容。非结构化数据的例子有社交媒体的内容、图片、音频、视频、各种文档和 PDF 文件。

2. 关系数据库管理系统

关系数据库管理系统用于处理结构化数据，是一种基于关系模型的、拥有明确定义的数据操纵和数据查询功能的数据库管理系统。在表 1-1 列出了排名前十的数据库引擎，其中有 7 种是关系数据库，说明这是数据库的主流类型。

SQL（Structured Query Language，结构化查询语言）是一种实现关系操作的语言，它是基于严格的数学理论的，因此关系数据库是有坚实的数学理论基础的。学会了 SQL，就可以很容易地使用其他各种关系数据库管理系统。

SQL 是在 20 世纪 70 年代随着关系数据库的出现而产生的，1986 年 ANSI 将其制订为标准 SQL-86，随后 ISO 组织也采用这个标准，称其为 SQL-87。SQL 经过了多次修订，其标准化历程如表 1-6 所示。

表 1-6 SQL 的标准化历程

年份	ANSI 标准	别名	说明
1986	SQL-86	SQL-87	ANSI SQL 的最初版本
1989	SQL-89	FIPS 127-1	少量修订
1992	SQL-92	SQL2	重要修订，是一个标志性的标准
1999	SQL:1999	SQL3	增加了正则表达式、递归查询、触发器、过程控制流等
2003	SQL:2003	SQL 2003	引入 XML 支持、标准化序列、自动生成 ID
2006	SQL:2006	SQL 2006	提供了对 XML 更多的支持
2008	SQL:2008	SQL 2008	引入了 Truncate 等

2011	SQL:2011	SQL 2011	引入了时序数据等
2016	SQL:2016	SQL 2016	引入了 JSON 等

SQL 是一种国际标准，是一种非常成熟的语言，一般所说的支持 SQL 标准通常是指支持 SQL-92 或 SQL-99 标准，所有关系数据库管理系统都是基于 SQL 的。

典型的关系数据库管理系统有 MySQL、SQL Server、Oracle 和 SQLite 等，在关系数据库管理系统上开发出来的应用系统的例子有财务系统、销售系统、银行系统，以及各种管理系统等等。

3. 非关系数据库管理系统

关系数据库用于处理结构化的数据，对于半结构化、非结构化数据，就需要用非关系数据库来处理。每一种非关系数据库都有各自的特点，适合不同种类的数据以及不同的处理要求。

非关系数据库管理系统（Not Only SQL，NoSQL，意为“不仅仅是 SQL”）适用于处理各种结构的数据（结构化、半结构化和非结构化），包括长文本、图像、音频、视频，用于满足新的业务需求。

典型的非关系数据库管理系统有 MongoDB、Redis 和 Elasticsearch 等，根据所处理数据的特点的不同和所采用技术的不同，还可细分为多种类型，例如表 1-1 中“说明”所述的三种类型，文档存储数据库、键值（Key-Value）存储数据库和全文搜索引擎数据库。

1.5.5 数据库与大数据和人工智能

大数据和人工智能是与数据库密切相关的信息处理技术，数据库技术是基础，大数据和人工智能是未来的发展方向。

数据库技术是数据管理的基础，它提供了数据的存储、查询、更新和管理等功能。随着数据量的不断增大，数据库技术也在不断发展，从关系型数据库到非关系型数据库，从结构化数据到非结构化数据，数据库技术都在不断地适应和满足数据管理的需求。

大数据是指无法用常规软件在短时间内处理的大量复杂数据集合。大数据技术包括数据采集、存储、处理、分析和可视化等方面，它能够帮助企业从海量数据中挖掘出有价值的信息，为企业决策提供支持。大数据技术的发展离不开数据库技术的支持，数据库技术为大数据提供了数据存储和管理的基础。

人工智能是指通过计算机程序模拟人类智能的技术，包括机器学习、深度学习、自然语言处理等方面。人工智能技术的发展离不开大数据的支持，大数据为人工智能提供了海量的训练数据和测试数据，使得人工智能系统能够不断地优化和改进。同时，数据库技术也为人工智能提供了数据存储和管理的支持。

【实战演练】“Goods 数据库”的开发

任务 1 MySQL 的下载、安装和配置

在一台计算机上，例如在自己的计算机上，安装和配置 MySQL 8.0.36，包括 MySQL Workbench。

任务 2 “Goods 数据库”项目



参考“1.3.3 使用 MySQL”的“Hello，数据库”项目，开发一个“Goods 数据库”项目，创建一个名为 goods 的数据库，将如图 1-32 所示的数据保存到这个数据库中。

	A	B	C	D	E	F
1	商品名	品牌	规格	计量单位	零售价格	商品分类
2	微波炉	格兰仕	20L	台	279	家电
3	洗衣机	美的	10KG	台	1590	家电
4	电热水壶	小米	1.7L	只	89	家电

图 1-32 商品信息

要求如下。

- 分析和设计：先分析图 1-32 所示的数据，然后设计好表名、列名和数据类型。
- 创建数据库：在 MySQL Workbench 上创建名为 goods 的数据库。
- 创建表：根据自己设计好的表名、列名和数据类型等，创建表。
- 输入数据：输入图 1-32 所示的数据。
- 查询数据：尝试编写一条 Select 语句，查询得到如图 1-32 所示的数据。

【单元重点】

单元小结参见本单元的【思维导图】，单元重点如下。

- 数据库、数据库管理系统、数据库系统等概念。
- 常见的四种数据库管理系统：MySQL、SQL Server、Oracle 和 SQLite。
- MySQL 8.0 的安装和配置。
- MySQL 的使用，通过 MySQL 客户端（MySQL Workbench）来操作 MySQL 服务器，具体的操作有创建数据库、创建表、输入数据和查询数据，需要熟练掌握。
- 主键（PK，Primary Key）是每一行数据的唯一标识，它的值不能重复，不能为空（Not Null），通常是自动增量的。
- 数据库管理系统的四大功能：数据定义、数据操纵、数据查询和数据控制。

【课后思考】

一、选择题

1. 数据库系统的核心是（　　）。

   A. 数据　　B. 数据库　　C. 数据库管理员　　D. 数据库管理系统

2. 数据库管理系统是（　　）。

   A. 一种操作系统　　B. 一种系统软件　　C. 一种图形软件　　D. 一种计算机语言

3. 数据库管理系统具有【　　　　】的功能。

   A. 数据定义　　B. 数据操纵　　C. 数据查询　　D. 数据获取

4. MySQL 是一种（　　）。

   A. 层次数据　　B. 网状数据库　　C. 关系数据库　　D. 面向对象数据库

5. 主键（PK，Primary Key）的必备特点是【　　　　】。

   A. 必须唯一　　B. 不能为空　　C. 自动生成　　D. 连续排列

6. SQL 是一种（　　）。

   A. 操作系统　　B. 系统软件　　C. 图形软件　　D. 计算机语言

二、填空题

1. 本单元安装的两个软件是____________________和____________________。
2. MySQL 中具有最高权限的系统管理员的账号是__________。

三、思考题

3. 数据库、表之间是什么关系？
4. 可以从哪几个方面定义一个列？
5. 什么是 PK？它的地位和作用是什么？

【课外拓展】

1、 问一问 AI（自行选择一个 AI 平台，或其他途径），关于 SQL（Structured Query Language）的知识，例如“什么是 SQL”、“SQL 的基本概念”、“SQL 的基本操作”或“SQL 的发展历史”等，增进对 SQL 的理解。
2、 问一问 AI，关于 MongoDB 数据库的知识，并比较它与 MySQL 的区别。
3、 问一问 AI，了解一些关于数据库的 B/S 架构、客户/服务器、单用户、主从式、分布式的知识。

单元2 理解关系数据库

【学习目标】

知识目标

- ◆ 了解数据库的开发过程。
- ◆ 理解数据模型的 3 要素。
- ◆ 理解 ER 模型和关系模型。
- ◆ 深刻理解并切实掌握实体、实体间联系的概念。
- ◆ 深刻理解主键和外键、主表和从表的概念。
- ◆ 理解关系中的 4 种异常。
- ◆ 理解用规范化设计来消除 4 种异常。
- ◆ 深刻理解“好”的关系数据库的要求。
- ◆ 掌握在范式理论的指导下进行关系数据库设计。

能力目标

- ◆ 学会 ER 模型向关系模型的转换。
- ◆ 学会使用规范化设计方法进行设计。

素质目标

- ◆ 提高逻辑思维能力。
- ◆ 树立全局观念，培养道德规范。

【思维导图】



【情景导入】

小明对数据库有了初步的了解后，觉得数据库并不难，也很有兴趣继续学习，因此他听从了学长的建议，先学习一些关系数据库的基础知识，学习如何设计数据库，打好数据库的基础。现在让我们与小明一起学习吧。

【知识储备】

MySQL 是一种关系数据库，因此，首先讲解一些关系数据库的基础知识。在讲解关系数据库基础知识之前，先简单介绍一下数据库的开发过程，以便读者对此有一个全局的了解，然后通过图书信息项目体验一下关系数据库，再讲解数据模型、ER 模型、关系模型和关系数据库设计，最后运用所讲解的知识，设计一个书店管理项目。

2.1 数据库开发过程

数据库开发是一个复杂的过程，大体上可以分为需求分析、数据库设计、应用程序开发和运行维护阶段，具体来说，可以分为下述六个阶段。

1. 需求分析阶段

需求分析是数据库开发的起点，主要任务是调查、收集与分析用户在数据处理中的数据需求、功能需求、完整性和安全性需求。经过反复修改和用户的确认，最终形成需求分析报告，即软件需求规格说明书，这是项目设计和开发的最重要的依据，也是项目验收的核心依据。

如果需求分析不完整，甚至是出现了错误，就会使开发出的项目达不到用户的实际需求，导致项目开发的失败。如果在开发进行的过程中，需要补充或变更需求，将会使开发的工作量成倍增长，导致项目开发的延期或失败。

因此，需求分析是决定一个项目成败最重要的一个环节。

2. 概念设计阶段

以需求分析的结果为依据，将客观事物及其联系抽象为实体及其属性、实体间的联系，从而建立概念数据模型。



概念设计是在不考虑任何物理因素（如软硬件平台、编程语言、DBMS 等）的情况下，进行数据建模的过程。

3. 逻辑设计阶段

将上一阶段得到的概念模型转换成关系模型，成为程序员能够理解和实施的模型，这样的模型就是逻辑数据模型。



逻辑设计是在不考虑具体 DBMS 选型的情况下，根据关系模型的要求进行数据建模的过程。通常是在概念模型的基础上构建逻辑模型。

4. 物理设计阶段

根据 DBMS 特点和处理的需要，对逻辑设计得到的关系模型进行物理设计，使其能够在具体的 DBMS 上实施。具体的内容包括表的结构、列的数据类型、数据完整性约束（主键约束、外键约束、唯一性约束和非空约束等）、视图和索引等，成为在计算机上能够运行的模型。



物理设计是在选定的 DBMS（如 MySQL）的情况下，进行数据建模的过程。通常是在逻辑模型的基础上构建物理模型。

5. 应用开发阶段

根据概念设计、逻辑设计的结果，特别是物理设计的结果，创建数据库、表等数据库对象，还要根据用户需求，编写数据操纵和查询语句、建立索引，编写存储过程和存储函数等。

在应用系统开发过程中，要将 DBMS 提供的 SQL 语句嵌入在程序设计语言中，例如 Java 或 PHP 语

言，为用户提供良好的界面，对数据库进行增删改，以及查询操作。

6. 运行维护阶段

数据库应用系统开发调试完成后，要在服务器上安装部署，才能投入正式运行。这时，用户通过网络访问和使用服务器上的数据库应用系统，进行输入数据、更新数据、删除数据、查询数据等操作，并确保数据库正常运行，一个数据库项目通常需要正常运行数年至数十年。

在运行过程中还需要维护，维护任务包括保证数据库的安全，防止数据泄露和攻击，定期备份数据库，在需要时（例如数据意外损毁后）恢复数据库中的数据，以及对数据库的性能监视、分析和改进，并不断地进行评估、调整与优化。

本单元讲解前 4 个阶段需求分析、概念设计、逻辑设计和物理设计阶段。单元 3~单元 7 讲解第 5 阶段应用开发阶段，单元 8 讲解第 6 阶段运行维护阶段。

2.2 体验关系数据库

本节通过图书信息数据库来体验关系数据库的设计和实施，了解数据库应用开发的流程，包括从数据库的需求分析、数据库设计、创建数据库和表、数据输入，到数据查询等整个过程。

2.2.1 图书信息数据库的分析

在进行数据库操作之前，首先需要调查清楚项目的需求是什么，需要处理什么样的数据，这些数据有什么特点。

1. 需求描述

1) 项目概况

项目名称：图书信息数据库

目标用户：出版社

2) 需求概述

本项目是一个面向出版社的图书信息发布系统，供各个出版社向销售商发布新书信息。

2. 收集数据

经过调研，从出版社收集到的发布信息例子数据如图 2-1 所示，这些数据原来是保存在 Excel 文件中的。

	F	G	H	I	J	K	L	M
1	作者	书名	出版年份	ISBN书号	价格	出版社	地址	联系电话
2	黄能耿	MySQL数据库应用实战教程	2022	9787115563798	59.8	人民邮电出版社	北京市丰台区成寿寺路11号	010-81055256
3	周德伟、覃国蓉、任仙怡	MySQL数据库基础实例教程	2021	9787115564634	49.8	人民邮电出版社	北京市丰台区成寿寺路11号	010-81055256
4	黄能耿、胡丽丹	Java EE应用开发及实训	2022	9787111687542	69	机械工业出版社	北京市百万庄大街22号	010-88361066

图 2-1 图书信息的例子数据

2.2.2 图书信息数据库的设计

1. 数据库设计思路

对于图 2-1 所示的数据，可以有两种设计思路，一种办法是把所有数据都保存在一张比较大的表中，另一种办法是把出版社和图书的数据分别保存在出版社表和图书表中。下面分别讨论这两种办法。

1) 单表方案

在设计上，单表方案是最简单的，在数据库中设计一个与图 2-1 完全相同的表，还需要加上主键，然后输入数据并发布，这样的设计思路与单元 1 的“Hello，数据库”的 book_list 表的设计思路相同。

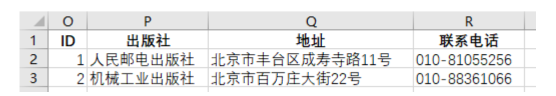
这个方案有如下缺点。

- 每次输入一本图书的信息时，都要输入出版社的信息，导致出版社信息的重复输入，增加了输入的工作量，也浪费了计算机的存储空间。
- 在输入数据之后再要修改出版社的数据，例如出版社的联系电话，这时要找到这个出版社的所有图书，修改每种图书的联系电话，这个工作量也是巨大的。
- 重复输入或重复修改出版社信息，不仅增加了输入工作量，还容易导致出错，因为不能保证每次输入的同一出版社的信息是相同的。

因此，需要一种解决方案，避免重复输入出版社信息，减少输入工作量，消除输入错误。

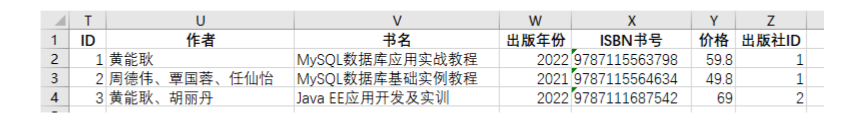
2) 双表方案

双表方案是把出版社和图书的数据分别保存在出版社表和图书表中，因此，图 2-1 所示的数据将成为如图 2-2 和图 2-3 所示的两张表。



	O	P	Q	R
1	ID	出版社	地址	联系电话
2	1	人民邮电出版社	北京市丰台区成寿寺路11号	010-81055256
3	2	机械工业出版社	北京市百万庄大街22号	010-88361066

图 2-2 出版社的例子数据



	T	U	V	W	X	Y	Z
1	ID	作者	书名	出版年份	ISBN书号	价格	出版社ID
2	1	黄能耿	MySQL数据库应用实战教程	2022	9787115563798	59.8	1
3	2	周德伟、覃国蓉、任仙怡	MySQL数据库基础实例教程	2021	9787115564634	49.8	1
4	3	黄能耿、胡丽丹	Java EE应用开发及实训	2022	9787111687542	69	2

图 2-3 图书的例子数据

输入数据时，先在出版社表中输入出版社数据，然后在输入图书数据时，有关出版社的信息是通过“出版社 ID”参照（引用）出版社表的主键“ID”值来实现的，这样就避免了数据的重复输入。

双表方案完美地解决了单表方案的缺点，因此双表方案的优点如下所示。

- 不需要重复输入出版社的信息，只需输入一次，然后被参照（引用）。
- 修改出版社信息时，只需在出版社表上修改，并且与图书表无关。
- 由于无需重复输入或重复修改出版社信息，不仅降低输入工作量，还避免了出错。

图书表的“出版社 ID”列有特别重要的意义，这种列称为外键（FK，Foreign Key），外键的值必须是主键中已经存在的值。在图 2-2 和图 2-3 所示的例子中，图书表的出版社 ID 的值就只能是在 1 和 2 之间选一个，因为出版社表的主键值只有 1 和 2。

在这个例子中，主键 ID 体现了它的价值，因为 ID 是唯一的并且不为空，当图书表的“出版社 ID”参照出版社的主键时，就能找到唯一的一个出版社，而不可能找不到，或者找到多个。

2. 数据库设计

下面按照双表方案对数据库进行设计，根据项目和表的含义进行如下命名。

- 数据库名：bookinfo
- 表名：publisher 和 book

表 2-1 是根据图 2-2 的数据，对出版社表（publisher）所作的设计。主键列建议命名为 id，其他列的列名前加上 col_作为前缀，目的是避免与 MySQL 的关键字冲突。

表 2-1 出版社表（publisher）的结构

列标题	要求	数据类型	建议的列名
主键（ID）	整数，不允许重复，自动编号	int	id
出版社	可变长字符串，最多 45 个字符	varchar(45)	col_name

地址	可变长字符串，最多 100 个字符	varchar(100)	col_addr
联系电话	可变长字符串，最多 45 个字符	varchar(45)	col_tel

图 2-2 是根据图 2-3 的数据，对图书表（book）所作的设计。其中 year 是 MySQL 中用于表示年份的一种数据类型，year 是 MySQL 的一个关键字，将出版年份命名为 col_year，避免了与其产生冲突。

外键列的命名建议用 id_作为前缀，后接被参照的表的表名，如表 2-2 最后一行所示。

表 2-2 图书表（book）的结构

列标题	要求	数据类型	建议的列名
主键（ID）	整数，不允许重复，自动编号	int	id
作者	可变长字符串，最多 45 个字符	varchar(45)	col_author
书名	可变长字符串，最多 100 个字符	varchar(100)	col_title
出版年份	年份，范围是 1901-2155	year	col_year
ISBN 书号	可变长字符串，最多 45 个字符	varchar(45)	col_isbn
价格	实数，两位小数	decimal(9,2)	col_price
外键（出版社 ID）	整数，参照出版社表的主键	int	id_publisher

2.2.3 图书信息数据库的开发



在数据库设计完成之后，就可以使用 MySQL Workbench 进行开发。读者可以在实训 2-1 的指导下完成下面的步骤（见附录 C）。

1. 创建数据库

在 MySQL Workbench 中创建数据库，数据库名称是 bookinfo。

操作过程参见单元 1 的图 1-19 及其说明。

2. 创建表

根据表 2-1 和表 2-2 的设计，在数据库 bookinfo 中创建两张表，表名分别是 publisher 和 book。

操作过程参见单元 1 的图 1-22 及其说明。

3. 输入数据

分别向出版社表（publisher）和图书表（book）输入如图 2-2 和图 2-3 所示的数据。

操作过程参见单元 1 的图 1-25 及其说明。

输入完成后，出版社表（publisher）和图书表（book）中的数据如图 2-4 所示。



id	col_name	col_addr	col_tel
1	人民邮电出版社2323424	北京市丰台区成寿寺路11号	010-81055256
2	机械工业出版社	北京市百万庄大街22号	010-88361066
NULL	NULL	NULL	NULL

id	col_author	col_title	col_year	col_isbn	col_price	id_publisher
1	黄能耿	MySQL数据库应用实战教程	2022	9787115563798	59.80	1
2	周德伟、覃国...	MySQL数据库基础实例教程	2021	9787115564634	49.80	1
3	黄能耿、胡丽丹	Java EE应用开发及实训	2022	9787111687542	69.00	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 2-4 数据库中出版社表（publisher）和图书表（book）的数据

4. 查询数据

打开“SQL File”选项卡，输入下述查询语句，操作过程参见单元 1 的图 1-28 及其说明。

```
Select col_author 作者, col_title 书名,
        col_year 出版年份, col_isbn ISBN 书号,
        col_price 价格, col_name 出版社,
        col_addr 地址, col_tel 联系电话
    from book join publisher
        on book.id_publisher = publisher.id;
```

运行结果如图 2-5 所示，与图 2-1 所示的数据比较，两者是完全相同的。

作者	书名	出版年份	ISBN书号	价格	出版社	地址	联系电话
黄能耿	MySQL数据库应用实战教程	2022	9787115563798	59.80	人民邮电出版社	北京市丰台区成寿寺路11号	010-81055256
周德伟、覃国蓉、任仙怡	MySQL数据库基础实例教程	2021	9787115564634	49.80	人民邮电出版社	北京市丰台区成寿寺路11号	010-81055256
黄能耿、胡丽丹	Java EE应用开发及实训	2022	9787111687542	69.00	机械工业出版社	北京市百万庄大街22号	010-88361066

图 2-5 查询结果

在数据库中，数据是分别保存在两张表中的（出版社表 publisher 和图书表 book，如图 2-4 所示），但是查询语句可以将两张表中的数据连接起来，显示在一起，起连接作用的就是图书的外键（出版社 ID）参照出版社的主键（id）。



在执行查询语句前，不要忘记双击数据库名 bookinfo，使其名字加粗显示，成为当前打开的数据库，否则会出现找不到 book 表和 publisher 表，因为在其他数据库中没有这两张表。

5. 在浏览器查看项目结果



与单元 1 的“Hello，数据库”一样，读者也可以从浏览器打开项目网站。根据附录 D 的说明，打开项目 2a “图书信息数据库”项目，如图 2-6 所示。



图 2-6 图书信息数据库项目的应用程序界面

这个界面与单元 1 的“Hello, 数据库”的界面有些不同，上半部分是出版社表（也称为出版社实体），下半部分是图书表（也称为图书实体）。用鼠标选择出版社时，下方就会列出该出版社发行的图书，一家出版社可以发行多种图书，而一种图书只能由一家出版社发行，这种联系称为一对多联系。

注意观察下方图书的“出版社 ID”（外键）的值就是上方出版社的“ID”（主键）的值，外键参照主键，意思是“图书的外键”等于“出版社的主键”。

2.2.4 图书信息数据库的特点

前面三个小节对图书信息数据库进行了分析和设计，并且在 MySQL 上完成了这个小项目。这个数据库有如下特点。

- 一个数据库可以由多张表组成，这个例子是两张表。
- 每张表描述一组独立的信息，这个例子是出版社和图书这两组独立的信息。
- 每张表都有一个主键，主键的值必须唯一，不允许为空（即不能没有值）。
- 一张表可以有外键，外键的值必须是另一张表主键的值中的一个。这个例子是“出版社 ID”。
- 表与表之间可以有联系，这个联系是通过外键参照主键而建立的，这个例子的两张表有一对多

联系。

- 在查询时，可以将两张表或多张表的信息连接在一起，如图 2-5 所示。

2.3 数据库基础知识

上一小节讨论了图书信息数据库的几个特点，这些特点是关系数据库最重要的特点，记住这几个特点对学习和理解关系数据库基础有极大的帮助。

下面讲解数据库的基础知识，从数据模型开始讲起。

2.3.1 数据模型

数据模型是对现实世界数据关系的抽象，用来描述数据、组织数据和对数据进行操作。

1. 数据模型三要素

数据模型所描述的内容包括三个要素：数据结构、数据操作及数据完整性约束，以保证数据库中数据的完整性、一致性和可靠性。

1) 数据结构

数据结构是对各种实体和实体间联系的表达和实现，实体是现实世界中客观存在的事物。

在上一节图书信息数据库项目的数据结构中，使用表这种结构来描述实体，每张表对应现实世界中的一个实体，即出版社表、图书表分别对应出版社实体、图书实体。

2) 数据操作

数据操作是对数据库中各种实体进行修改（插入、更新、删除）和检索（查询）等操作，数据模型必须定义这些操作的确切含义、操作符号、操作规则以及实现操作的语言。

在单元 1 和本单元，体验过了数据的插入和数据的查询，但还没有用过数据的更新和删除。

3) 数据完整性约束

数据完整性约束是对数据库中各种实体及其联系的约束性规定，用以保证数据库中数据的完整性、一致性和可靠性。数据完整性约束可以有效避免数据库中存在不符合语义规定的数据，防止因错误的数据造成无效操作或错误操作。

在上一节图书信息数据库项目中的主键和外键，对应了最重要的两种数据约束，即主键约束和外键约束，后面还会深入讲解。

2. 数据模型三层次

数据模型按不同的应用层次分成如下 3 种类型，这 3 种模型对应数据库开发过程中的 3 个阶段。

1) 概念数据模型

概念数据模型（Conceptual Data Model）对应概念设计阶段，是对现实世界的认识和抽象描述，从用户的角度对实体及其联系建立概念化的模型。常用的概念模型是 ER 模型，将在下一小节“2.3.2 ER 模型”讲解。

通俗地说，概念模型是从用户的角度来描述问题。

2) 逻辑数据模型

逻辑数据模型（Logical Data Model）对应逻辑设计阶段，是从计算机的角度，将概念模型转换为 DBMS 支持的某一种数据模型（如关系数据模型，将在“2.3.3 关系模型”讲解）。

通俗地说，逻辑模型是从程序员的角度来描述问题。

3) 物理数据模型（物理世界）

物理数据模型（Physical Data Model）对应物理设计阶段，是逻辑模型在具体的计算机系统（包括硬

件、软件和操作系统）上的实现，本书用 MySQL 设计的数据库都是物理数据模型。

通俗地说，物理模型是从计算机的角度来描述问题。

3. 几种典型的逻辑数据模型

按照数据模型三要素，特别是数据结构的不同，在数据库发展的历史上，产生过几种逻辑数据模型，其中有代表性的逻辑数据模型有如下四种，在单元 1“1.1.2 数据库的发展历史”中曾提及。

1) 层次模型

层次模型（Hierarchical Model）是最早出现的一种数据模型，它的数据结构是用树形结构来表示各类实体以及实体之间的联系，它的结构过于简单，常常无法完整地描述现实世界。层次模型主要流行于上世纪 60 年代。

2) 网状模型

网状模型（Network Model）是对层次模型的扩展，它的数据结构是网状的，从而导致结构复杂，难以扩充和维护。网状模型主要流行于上世纪 60~70 年代。

3) 关系模型

关系模型（Relational Model）用二维表结构来表示各类实体以及实体之间的联系。关系模型诞生于上世纪 70 年代，直到如今都是主流的数据库技术。

单元 1 的“Hello，数据库”和本单元的“图书信息数据库”采用的都是二维表，也都是关系模型，本书只讲解关系模型，以及基于关系模型的关系数据库。

4) 面向对象模型

面向对象模型（Object Oriented Model）是采用面向对象技术进行数据建模的一种技术。面向对象模型诞生于上世纪 70 年代，是一种有发展潜力的数据模型，例如单元 1 数据库引擎排行榜提到的排名第 4 的 PostgreSQL 就是一种同时具有关系模型和面向对象模型特征的数据库管理系统。

2.3.2 ER 模型

实体联系模型（Entity-Relationship Model，ER 模型）是一种在概念设计阶段使用的数据模型构建工具，然后在逻辑设计阶段再转换为关系模型。

1. 常用术语

下面先介绍一些与概念模型有关的常用术语，有些术语在前面的讲解中已经使用过，在这里给出准确的定义，以便更好地理解 ER 模型。

1) 实体（Entity）

客观存在并可相互区别的事物称为实体，实体可以是具体的人、事、物或抽象的概念，例如图书信息数据库中的出版社是一个实体，图书也是一个实体，其他的例子如一位学生、一本书、一门课程、一份成绩等都是实体。

2) 实体集（Entity Set）

同一类型实体的集合称为实体集，例如分属各个班级的全体学生就是一个实体集。为简便起见，实体集常常简称为实体。

3) 属性（Attribute）

对实体特性的描述称为属性，例如描述学生实体的属性有学号、姓名、出生日期、性别等。

4) 属性值（Attribute value）

属性值是某个实体的属性的取值，例如“SW390105”“方博涵”“2004-11-15”和“女”分别是方博涵

这个学生实体的学号、姓名、出生日期、性别属性的属性值。

5) 域（Domain）

属性值的取值范围称为域。如学号域为 2 个字母加 6 个数字的字符串（不同学校对学号的规定可以不同），性别域为“男”和“女”。

6) 键（Key）

键是实体的某个属性或属性集，用于确保数据的完整性、唯一性和关联性，例如图书表（book）中的“ISBN 书号”属性，它能够唯一标识一本书。在下一小节的关系模型中第 3 部分“候选键、主键和外键”还要作深入的讲解。

2. ER 图

ER 模型的基本建模元素是实体、属性和联系，表现形式是实体联系图（Entity-Relationship Diagram，ER 图），分别使用下述三种符号表示实体、属性和联系。

- ▭：用矩形表示实体。
- ⬭：用椭圆表示属性，并用无箭头直线标出实体与属性的关系。
- ◇：用菱形表示实体间的联系，并用无箭头直线标出实体间的联系，可选地加上联系的类型。

作为 ER 模型的一个例子，图 2-7 是图书信息数据库对应的 ER 图。

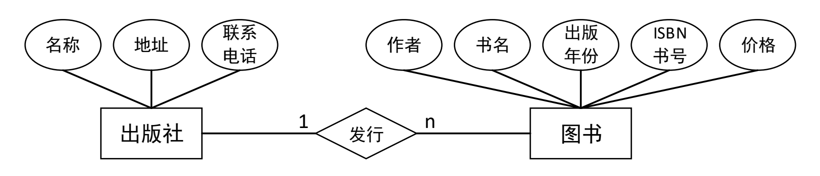


图 2-7 图书信息数据库的 ER 图

图 2-7 所示的 ER 图有两个实体：出版社实体和图书实体，每个实体都有一些属性，如名称、地址、书名、价格等，出版社实体和图书实体有“发行”的联系，出版社“发行”图书，这个联系是一对多的联系，出版社是一的一方（图中用 1 表示），图书是多的一方（图中用 n 表示），即一个出版社发行多种图书，而一种图书只由一个出版社发行。

3. 实体的联系

两个实体之间的联系（Relationship）可以分为三种类型：一对一联系、一对多联系和多对多联系，如表 2-3 所示。

表 2-3 三种联系类型

联系类型	定义	例子
一对一联系（记为 1:1）	若实体集 A 中每个实体只能与实体集 B 中的一个实体有联系，反之亦然，则称实体集 A 与实体集 B 存在一对一的联系。	例如一个班级只有一个班主任，一个班主任只能管理一个班级，这时班级实体集与班主任实体集是一对一的联系。如果一个班主任可以管理多个班级，这时就不是一对一的联系。
一对多联系（记为 1:n）	若实体集 A 中每个实体与实体集 B 中的任意多个实体有联系，反过来实体集 B 中每个实体至多与实体集 A 中的一个实体有联系，则称实体集 A 与实体集 B 存在一对多的联系。	例如班级实体集与学生实体集是一对多的联系，因为一个班级有多个学生，而一个学生只能属于一个班级，如图 2-8 所示。
多对多联系（记为 m:n）	若实体集 A 中每个实体与实体集 B 中的任意多个实体有联系，反过来实体集 B 中每个实体与实体集 A 中的任意多个实体有联系，则称实体集 A 与实体集 B 存在多对多的联系。	例如学生实体集与课程实体集是多对多的联系，因为一个学生可以选修多门课程，同样一门课程可以有多个学生选修，如图 2-8 所示。

联系可以像实体一样拥有属性。例如，如图 2-8 所示，学生与课程之间有一个“选修”的联系，拥有“课程号”、“学号”和“成绩”等属性。

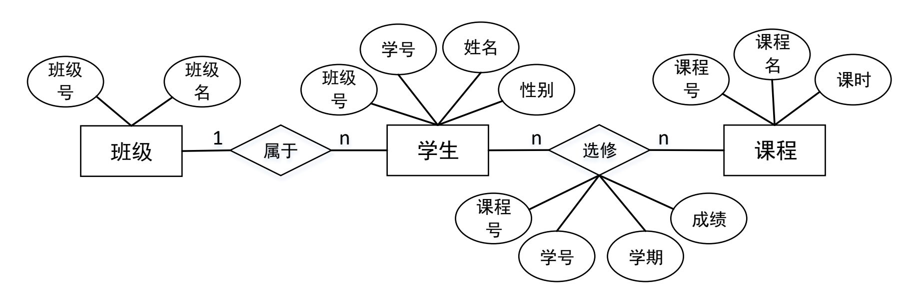


图 2-8 班级、学生和课程的 ER 图

还有一种联系是多对一联系，多对一联系和一对多联系是对同一个联系的不同表述，如果 A 和 B 是多对一联系，那么 B 和 A 就必定是一对多联系，因此一对多联系和多对一联系是互为镜像的。例如，出版社与图书是一对多的联系，一个出版社可以发行多种图书，但一种图书只由一家出版社发行（当不考虑联合发行时），反过来，图书与出版社是多对一的联系，这是很容易理解的。通常很少提多对一联系这种情况，仅在需要特别强调时才会提到。

4. ER 模型实例

这里以一个书店管理项目为例，创建一个 ER 模型，采用 ER 图表示。

书店管理的实体主要是图书实体和客户实体，就是将图书卖给客户，在销售的过程中，需要有一个订单实体，这是类似于发票或销售小票的单据。为方便输入出版社数据，还要有一个出版社实体，如图 2-9 所示，每个实体都有相应的属性。

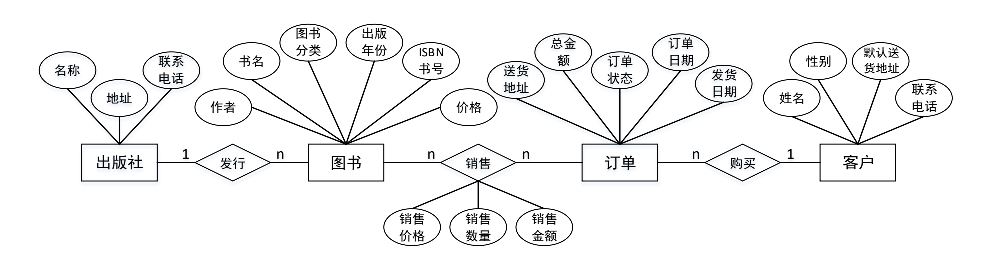


图 2-9 书店管理的 ER 图

客户可以多次下单，客户与订单是一对多联系，出版社与图书也是一对多联系。而图书与订单则是多对多联系，一个订单可以订购多种图书，而一种图书也可以销售给多个订单，并且这个联系还拥有属性，记录每本图书的销售数量，如果有销售折扣，还要记录销售价格和销售金额。

2.3.3 关系模型

关系模型（Relational Model）是一种在逻辑设计阶段使用的数据模型构建工具，它是关系数据库的基础，1970 年代由 IBM 公司的 E. F. Codd 博士等提出。Codd 博士被誉为“关系数据库之父”，由于他的杰出贡献，于 1981 年获得 ACM 图灵奖（图灵奖是计算机界的最高奖项，相当于计算机界的诺贝尔奖）。从 1980 年代开始，关系模型取代了网状模型和层次模型，成为应用最为广泛的数据库技术。

1. 关系的定义

关系（Relation）是满足一定条件的二维表，表中的行称为元组，表中的列称为属性。需要满足的条件是关系的 6 项基本特征，如表 2-4 所示。

表 2-4 关系的 6 项基本特征

特征	说明
行（元组）的唯一性	关系的行（元组）不能重复，即关系必须有一个主键，用于唯一标识行（实体）。
列（属性）的唯一性	关系的列（属性）不能重复，即列名（属性名）不能重复。
行（元组）的次序无关性	关系的行（元组）的次序是无关紧要的，行的次序仅仅在输出时有意义。
列（属性）的次序无关性	关系的列（属性）的次序是无关紧要的，列的次序仅仅在输出时有意义。
值域的统一性	列（属性）的值必须属于同一个域，例如性别列的值只能是“男”或者是“女”。
列（属性）的原子性	列（属性）是不可分割的最小数据项，即属性必须满足原子性的要求。

表 2-4 中 6 项基本特征中的前 5 项比较容易理解，这里解释一下最后一项特征“列的原子性”。例如图 2-10 所示的温度属性不是原子性的，因为温度属性分割为两个子属性：最高温度和最低温度，但是可以有两个独立的属性：最高温度和最低温度，这时两个属性都是原子性的，见图 2-11。

序号	日期和时间	观测点	温度		风速
			最高温度	最低温度	

图 2-10 “温度”属性不是原子性的

序号	日期和时间	观测点	最高温度	最低温度	风速

图 2-11 所有属性都是原子性的



属性不是原子性的表（如图 2-10）也可以认为是三维表，因此可以用二维表这个术语来强调表的所有属性都是原子性的。

关系的每一行定义一个实体，每一列定义实体的一个属性。因此一个关系（二维表）就是一个实体集。例如图书信息数据库中的出版社表和图书表都是关系，同时也是实体。

对同一个概念常常有不同的名称，原因是对同一个概念由不同领域的专家命名就有不同的名称，如表 2-5 所示。对于实体集、实体和属性，数学家将其命名为关系、元组和数据项，程序员将其命名为数据表、记录和字段，后来为简单起见，将其命名为表、行和列。

表 2-5 实体集、实体和属性的同义词

概念模型（行业专家）	逻辑模型（数学家）	物理模型（老程序员）	物理模型（新程序员）
实体集 entity set	关系 relation	数据表 data table	表 table
实体 entity	元组 tuple	记录 record	行 row
属性 attribute	属性 attribute 或数据项 data item	字段 field	列 column

例如，单元 1“Hello，数据库”中的图书关系与各个术语的对应情况如图 2-12 所示。



图 2-12 图书关系

本书在不同场合可能使用不同的术语，用于表示不同的语境，例如当讨论 ER 模型（概念模型）时，使用实体集、实体和属性，当讨论关系模型（逻辑模型）时，使用关系、元组和属性（或数据项），在 MySQL（物理模型）里实际操作或编写 SQL 语句时，使用表、行和列。

2. 关系的表示

关系由关系名和属性组成，在数学上可以用下述方式表示关系。

关系名（属性名 1，属性名 2，…，属性名 n）

例如，图 2-12 所示的图书关系可以表示如下。

图书（id, author, title, price, publisher）

3. 候选键、主键、外键和非主属性

前一小节“3.2.3 ER 模型”的第 1 部分“常用术语”介绍过“键”的概念，在关系模型中，对“键”作了进一步的划分，对此再作深入的讨论。

1) 候选键（Alternate Key 或 Candidate Key）

能够唯一标识一个元组（实体）的属性或属性集称为候选键。例如在表示学生信息的二维表中，学号和身份证号都可以作为唯一标识学生的属性，因此这两个属性都是候选键，而学生姓名不能唯一标识一个学生，因为可能存在同名同姓的学生。

候选键也可能是由多个属性组成的属性集，例如一个成绩实体，拥有姓名、课程名和成绩三个属性，这时候选键是姓名和课程名所组成的属性集，因为姓名或课程名都不能单独唯一标识一个成绩，只有姓名和课程名的组合才能唯一标识一个成绩。

2) 主键（Primary Key）

在候选键中指定其中一个作为主要候选键，简称为主键。例如在学生表中，可以指定学号为主键，也可以指定身份证号为主键，但只能取其中之一。

在实际开发中，通常添加一个无业务含义的属性作为主键，其值由程序自动生成，这样的主键可以称为唯一标识（Identity，缩写为 id），在单元 1 和本单元的案例中都是这样做的。

3) 外键（Foreign Key）

关系中的某个属性或属性集虽然不是该关系的键，但却是另外一个关系的键，则称其为外键。换句话说，外键是本关系中的属性或属性集，它标识了另外一个关系中的实体。

在本单元的“图书信息数据库”中，就对图书表设计了一个外键。



主键唯一标识本表的一个实体（行）。外键标识其他表中的某个实体（行），这时，外键参照（引用）了其他表的实体（行）。

从理论上说，候选键可以是单属性的，也可以是多属性的（属性集），因此，主键和外键也可以是多属性的。但在实际设计时，都是采用无业务含义的唯一标识（id，单属性的）作为主键和外键。

4) 主属性（Prime Key）和非主属性（Non-prime Key）

包含在候选键中的属性称为主属性，因此，id、学号、身份证号等都是主属性，在前述的成绩实体中，候选键是姓名和课程名所组成的属性集，这时，姓名或课程名也是主属性。与之对应，不包含在任何候选键中的属性称为非主属性。



键也称为码，因此在有些资料上，候选键、主键、外键和非主属性称为候选码、主码、外码和非码属性。

4. 主表和从表

在两张有联系的表中，参照的表（外键所在的表）称为从表（或子表），被参照的表（被参照主键所在的表）称为主表（或父表）。

例如图书信息数据库中，图书表的外键参照出版社表的主键，因此这两张表有主从联系，图书表是从

表（子表，多的一方，儿子可以有多个），出版社表是主表（父表，一的一方，父亲只有一个），如图 2-13 所示，这是 ER 图的另一种画法，特别强调了表之间的参照联系，通常在物理设计阶段使用。

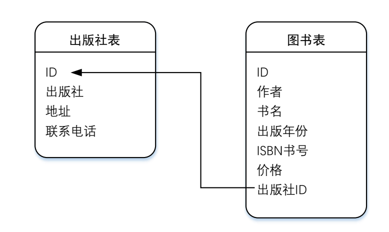


图 2-13 图书表的外键参照出版社表的主键，即从表参照主表

在图 2-13 中，图书表（从表）的外键参照出版社表（主表）的主键，用一条带箭头的折线表示参照联系，箭头的方向是从参照的表指向被参照的表，即从外键指向主键。这种参照称为图书表参照出版社表，即从表参照主表，或者是从表的外键参照主表的主键。



主表和从表是物理设计阶段的术语，对理解表之间的主从关系非常有用。但它们不是 ER 模型或关系模型的术语，没有对应的主实体或主关系这类术语。

5. 关系模型的三要素

关系模型的三要素是关系模型的数据结构、关系模型的数据操作和关系模型的数据完整性约束。

1) 关系模型的数据结构

关系模型的数据结构是二维表（即关系），见前述“关系的定义”，关系必须满足 6 项基本特征，参见表 2-4。

关系模型的数据结构非常简单，实体以及实体之间的联系都是用关系来表示。因此，可以用简单的模型来表示非常复杂的现实世界。

2) 关系模型的数据操作

关系模型的数据操作叫做关系操作，关系操作是对关系模型中各种对象进行修改（插入、删除、更新）和检索（查询）等的操作。在关系模型中，理论上是采用关系代数语言实现关系操作，实际编程中采用 SQL 语言进行关系操作（将在单元 4 和单元 5 讲解）。例如关系代数语言有投影π、选择σ、连接 R⋈S、并 R∪S、交 R∩S、差 R−S、除 R÷S、笛卡尔积 R×S 等操作，一些常用的关系操作与 SQL 语言的对应关系如表 2-6 所示。

表 2-6 关系操作与 SQL 语言的关系

功能	关系操作	对应的 SQL 语句
选择列	投影操作π	Select … from …
选择行	选择操作σ	Select * from … where …
内连接	连接操作 R⋈S	Select * from … inner join …
交叉连接	笛卡尔积 R×S	Select * from … cross join …
联合	并操作 R∪S	Select … union Select …

关系是元组的集合，对关系的运算就是对集合的运算，因此运算的结果是集合，这个集合也是关系。

关系模型以坚实的数学理论（关系代数、集合论和数理逻辑）为基础，可以对数据进行严格的定义、规范化和运算，这是关系数据库成为主流技术的根本原因。

3) 关系模型的数据完整性约束

关系模型的数据完整性约束分为三类，即实体完整性约束、参照完整性约束和用户定义完整性约束。

- 实体完整性约束（也称为主键约束）：是指任何一个关系必须有且只有一个主键，主键的值不能重复，也不能为空。简单来说，就是不允许存在一个缺少唯一标识的实体。
- 参照完整性约束（也称为外键约束）：是指外键的值可以为空或不为空，但其值必须是所参照的表的主键的值。简单来说，就是不允许参照一个不存在的实体，但是可以不参照。
- 用户定义完整性约束：这类约束反映了具体应用中的业务需求，例如学生的姓名不能为空（非空约束），学生的身份证号不允许重复（唯一性约束）。

数据完整性约束具有十分重要的意义，在本书中将多次提到，在单元 3“3.4 数据完整性约束”有详细讲解，在单元 4“4.5 数据操纵与数据完整性约束”还有具体的实例。

2.3.4 ER 模型向关系模型的转换

ER 模型是概念设计阶段建立的模型，关系模型是逻辑设计阶段建立的模型。在概念设计阶段向逻辑设计阶段转换的过程中，就需要将 ER 模型转换为关系模型。

ER 模型由实体、实体的属性、实体之间的联系三个建模元素组成，将 ER 模型转换为关系模型就是将实体、属性和联系分别转化为关系、属性以及主外键的参照。以下是转换的方法。

1. 实体的转换

实体可以直接转换为关系，转换的规则如下。

- 实体名转换为关系名。
- 实体的属性转换为关系的属性。
- 实体的键转换为关系的键。

2. 主键和外键

主键和外键是关系模型的概念。在 ER 模型中有键的概念，但是没有主键和外键的概念。

1) 主键

在关系模型中，每个关系都必须有主键，因此将实体转换为关系时，可以指定其中一个键为主键，但是通常的做法是为每个关系添加一个无业务含义的主键。

2) 外键

外键的作用是建立关系与关系之间的联系，因此在进行联系的转换时，需要为关系添加外键才能建立起关系间的联系，见下面的讲解。

3. 联系的转换

联系的转换需要根据联系的类型，采用不同的规则进行转换，如表 2-7 所示。

表 2-7 联系的转换规则

联系的转换	说明
一对多联系 转换为 主键和外键的参照	一对多联系在关系模型中表现为两个关系之间主键和外键的参照，多的一方的外键参照一的一方的主键，如图 2-14 所示。如果联系还拥有属性，可以将联系的属性合并到多的一方，成为多的一方的属性。
一对一联系的转换 转换为 主键和外键（*）的参照	一对一联系在关系模型中同样表现为两个关系之间主键和外键的参照，从属的一方的外键参照另一方的主键，并且要对外键（*）加上唯一性约束，如图 2-15 所示。如果联系还拥有属性，可以将联系的属性合并到任何一方，成为这一方的属性。
多对多联系	在多对多联系的情况下，联系也要转换为关系，成为一个新的关系，如果联系还拥有属性，则将

转换为 两个 1 对多联系	联系的属性转换为新关系的属性。新关系与原来两个关系形成两个多对一联系，原关系是一的一方，新关系是多的一方，如图 2-16 所示。新关系中有两个外键，这两个外键分别参照原来的两个关系的主键。

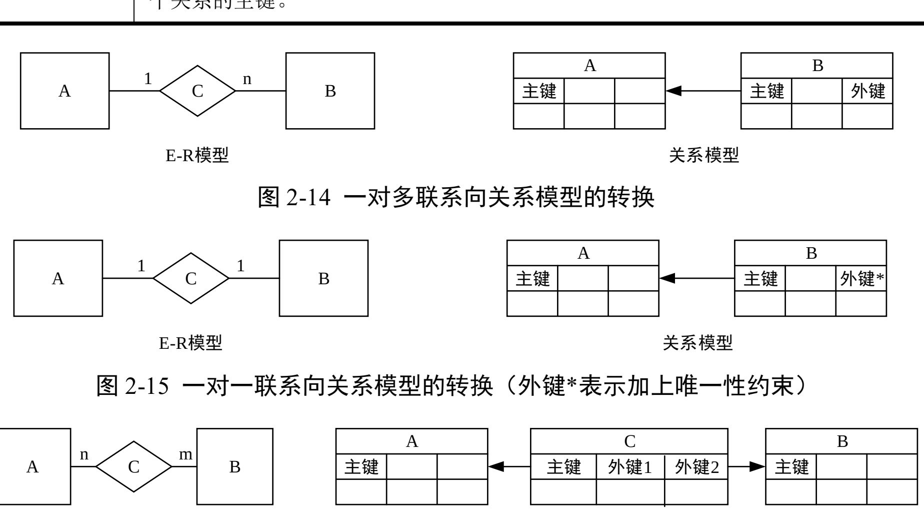


图 2-14 一对多联系向关系模型的转换

图 2-15 一对一联系向关系模型的转换（外键*表示加上唯一性约束）

图 2-16 多对多联系向关系模型的转换（联系也转换为关系）

4. 合并具有相同键的关系

具有相同键的关系表示的是同一个实体，因此应该合并，合并后的关系拥有原来两个关系的全部属性。

5. 转换的实例

1) 班级、学生和课程实例

例如，图 2-8 所示的 ER 模型可以转换为如下的关系，其中 3 个实体（班级、学生、课程）被转换为 3 个关系，一个多对多联系（选修）被转换为一个关系。

班级（班级号，班级名）
学生（学号，姓名，性别，*班级号*）
课程（课程号，课程名，课时）
选修（***学号***，***课程号***，学期，成绩）

上述关系中用底划线标示的属性是主键，用斜体标示的属性是外键，例如选修关系的主键是学号、课程号和学期三个属性组成的属性集，学号是一个外键，课程号是另一个外键。

对应的数据如图 2-17 所示，图中的箭头表示参照联系，方向是外键指向主键。

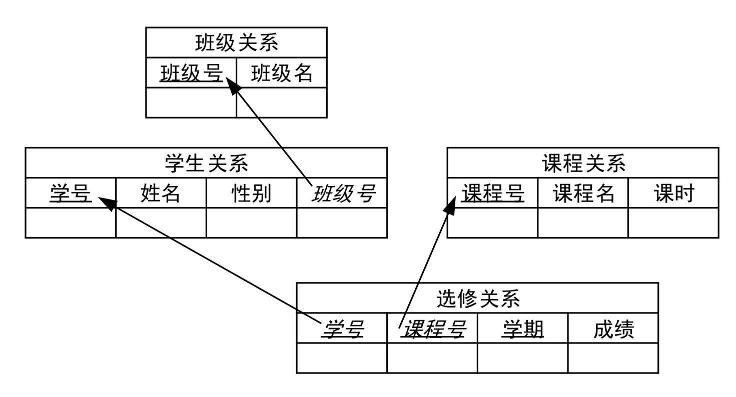


图 2-17 班级、学生和课程及其主外键的参照

图中班级关系的主键是班级号，学生关系的主键是学号，学生关系的外键班级号参照班级关系的主

键班级号，课程关系的主键是课程号，选修关系是由多对多联系转换而来，它的主键是学号、课程号和学期三个属性组成的属性集，并且学号和课程号还是外键，分别参照学生关系的主键学号和课程关系的主键课程号。

学期作为主键属性集的组成部分，是因为当考试成绩不及格时，用以区分另一个学期重修时的成绩。



打开附录 D“项目 2b “学生成绩管理”项目”项目，体验图 2-8 的学生与课程多对多“选修”联系转换为两个一对多联系后（参见图 2-17）的情况，向成绩表（由“选修”联系转换而来）添加成绩时，都要指定是哪一门课程和哪一位学生，并指定学期，如图 2-18 所示。

在项目 2b 中，为每张表都添加主键 id，原来的主键作为候选键，这是项目开发中的常规做法。



图 2-18 学生和课程的选修联系（多对多），拥有成绩属性

2) 出版社、图书、订单和客户实例

下面将图 2-9 所示的 ER 模型转换为关系模型，结果如下。

出版社（id，名称，地址，联系电话）
图书（id，作者，书名，图书分类，出版年份，ISBN 书号，价格，***出版社 ID***）
客户（id，姓名，性别，默认送货地址，联系电话）
订单（id，送货地址，总金额，订单状态，订单日期，发货日期，***客户 ID***）
订单明细（id，销售价格，销售数量，销售金额，***订单 ID***，***客户 ID***）

在图 2-9 所示的 ER 模型中，有的实体有键，如图书实体的键是 ISBN 书号，有的实体没有键，如订单实体是没有键的，有的实体的键是不严谨的，如客户实体的姓名，似乎可以唯一标识一个客户，但是可能存在同名同姓的客户。

在转换为关系模型时，为每张表添加主键 id，同时为一对多联系中多的一方加上外键。多对多联系“销售”转换为关系，并根据“销售”的具体含义，重新命名为“订单明细”，订单明细有两个外键分别参照订单的主键 id 和客户的主键 id。

2.4 关系数据库设计

关系数据库是有严密的数学理论基础的，因此关系数据库的设计也必须在数学理论的指导下进行，其中最重要的是范式理论，关系数据库设计就是在这个理论的指导下进行规范化设计，从而得到一个满足范式理论要求的数据结构。

在讲解范式理论之前，先讲解关系中的异常，而范式理论的主要目的就是消除关系中的异常。

2.4.1 关系中的异常

规范化设计的目标就是设计一个好的关系模型，因此，先分析一下设计得不好的关系中有哪些异常现象。



在“2.2.1 图书信息数据库的分析和设计”中讲解过两种设计思路“单表方案”和“双表方案”，读者可以回顾和对比一下。

下面通过一个例子来分析一个设计得不好的关系（即不满足规范化设计的要求）所存在的问题。考虑如表 2-8 所示的一个设计得不好的关系，这个关系的主键是学号。

表 2-8 设计得不好的“学生”关系

班级名称	班主任	班主任电话	<u>学号</u>	学生姓名	学生性别
软件 31431	李进中	13587654321	3143101	张三	男
软件 31431	李进中	13587654321	3143102	李四	女
软件 31431	李进中	13587654321	3143103	王五	男
网络 31451	汪一平	13712345678	3145101	赵六	男

表 2-8 所示的学生关系存在下述四个严重问题。

1. 数据冗余

在表 2-8 中，班主任李进中的名字和电话在数据中多次出现，这种现象称为数据冗余。冗余的数据会浪费大量的存储空间，并且也会降低数据库的运行效率，并导致下述三种异常的出现。

2. 更新异常

在表 2-8 中，当班主任李进中更换了电话号码，这时必须更新软件 31431 班所有学生的班主任电话，如果由于某种原因只更新了一部分，这时就出现了更新异常，如表 2-9 所示。

3. 删除异常

在表 2-8 中，如果删除了学生“赵六”，由于赵六是班上的最后一名学生，这时班级“软件 31432”和班主任“汪一萍”的信息就会随之消失，这是由于删除学生而导致意外删除了班级和教师，这时就出现了删除异常，如表 2-9 所示。

4. 插入异常

如果学校新来了一位教师“张明亮”，由于他还没有担任班主任，当插入这位教师的信息后，会引起班级为空，以及主键学号为空的情况，这时就出现了插入异常，如表 2-9 所示。

表 2-9 是通过数据来说明上述四种异常，说明了设计上的缺陷会导致数据库应用系统出现数据混乱，最终使数据库应用开发陷入失败的境地。

表 2-9 设计得不好的“学生”关系中的四种异常

班级名称	班主任	班主任电话	学号	学生姓名	学生性别
软件 31431	李进中【数据冗余】	12612344321	3143101	张三	男
软件 31431	李进中【数据冗余】	12612344321	3143102	李四	女
软件 31431	李进中【数据冗余】	13587654321【更新异常】	3143103	王五	男
~~网络 31451~~	~~汪一平~~	~~13712345678~~	~~3145101~~	~~赵六~~【删除异常】	男
	张明亮【插入异常】	13912341234	【主键为空】		

一个好的关系模型应该具备以下条件。

- 尽可能少的数据冗余，在数据库设计中，不可能没有数据冗余。
- 没有插入异常、删除异常和更新异常，这三种异常是不允许出现的。

2.4.2 规范化设计

规范化设计的好坏直接影响到数据库应用系统开发的成败。规范化设计的理论基础是范式理论，它是关系数据库设计的极其重要的基础。

1. 范式理论

数据库设计的范式是数据库设计所需要满足的规范，满足这些规范的数据库是简洁的、结构明晰的，并且不会发生插入异常、删除异常和更新异常，具有较低的数据冗余度。反之则是难以理解的，使数据库难以维护，将导致数据库项目开发的失败。

数据库范式（Normal Form，NF）有 1NF、2NF、3NF、BCNF、4NF 和 5NF 等，共计六级，范式级别越高，要求越严格。通常的规范化设计达到 3NF 或 BCNF 的要求即可，更高的范式级别（如 4NF 和 5NF）可能造成效率的降低，因此仅在必需时才使用。

2. 函数依赖

在讲解范式理论之前，先讲解函数依赖。函数依赖可分为完全函数依赖、部分函数依赖和传递函数依赖三种。

假设 X 为关系 R 中的某个属性或属性组，X'为 X 的任意非空子集；Y、Z 为关系 R 中的任意属性或属性组，并且 X、Y 和 Z 都互不包含。这时用符号“→”表示依赖（例如$A \to B$表示 B 依赖于 A，或者说 A 决定 B），用符号“↛”表示不依赖，则上述三类依赖关系的定义如表 2-10 所示。

表 2-10 完全依赖、部分依赖和传递依赖的定义

依赖的类型	定义和例子	表示法
完全函数依赖 full	定义：若$X \to Y$，$X' \nrightarrow Y$，则称 Y 完全函数依赖于 X	$X \xrightarrow{f} Y$
	以关系“选修（课程号，学号，成绩）”为例，成绩依赖于主键（课程号和学号），成绩不依赖于学号，也不依赖于课程号，这时，成绩是完全依赖于主键（学号和课程号）。	
部分函数依赖 partial	定义：若$X \to Y$，$X' \to Y$，则称 Y 部分函数依赖 X	$X \xrightarrow{p} Y$
	以关系“选修（课程号，学号，姓名，成绩）”为例，姓名依赖于主键（课程号和学号），但是姓名依赖于学号，不依赖于课程号，这时，姓名是部分依赖于主键（课程号和学号）。	
传递函数依赖 transitive	定义：若$X \to Y$，$Y \to Z$，且$Y \nrightarrow X$，则称 Z 传递函数依赖于 X	$X \xrightarrow{t} Z$
	以关系“选修（选修 ID，课程号，学号，姓名，成绩）”为例，姓名依赖于学号，学号又依赖于选修 ID，这时，姓名是传递依赖于主键（选修 ID）。	

下面分别讲解 1NF、2NF、3NF，其他几个范式因为不常使用，本书不予讲解。

3. 第一范式（1NF）

如果一个关系满足关系模型的 6 项基本特征（见表 2-4），并且属性的值只包含域中的一个单一的值，则称该关系属于第一范式（1NF）。就是说，属性值必须满足原子性的要求。



第一范式的要求是：属性值必须满足原子性的要求。而在关系的基本特征中有一个是：属性必须满足原子性的要求。注意前者是属性值，后者是属性。

例如表 2-11 中的“联系人”关系，李四的电话号码保存了两个值，违反了属性值原子性的要求，达不到 1NF 的要求。

表 2-11 违反“属性值原子性”的例子

主键	姓名	电话号码
1	张三	13512345671
2	李四	13512345672, 0510-12345678
3	王五	13512345673

表 2-12 违反属性值原子性的解决方案之一

主键	姓名	手机号码	固定电话
1	张三	13512345671	
2	李四	13512345672	0510-12345678
3	王五	13512345673	

解决的方案有如下两种。

方案一：拆分属性。将 “电话号码”属性拆分为两个属性“手机号码”和“固定电话”，用于分别保存两个值，如表 2-12 所示。

方案二：拆分关系。将“电话号码”从“联系人”关系中拆分出来，作为一个新的关系（需添加主键），命名为“联系方式”关系，新的关系“联系方式”作为多的一方添加一个外键，参照一的一方“联系人”的主键，如表 2-13 所示。联系方式还可以保存多种号码，如 QQ 号、微信号等，因此这种解决方案还提供了更多的功能。

表 2-13 违反属性值原子性的解决方案之二

“联系人”关系

主键	姓名
1	张三
2	李四
3	王五

“联系方式”关系

主键	联系方式	外键
1	13512345671	1
2	13512345672	2
3	0510-12345678	2
4	13512345673	3

4. 第二范式（2NF）

如果一个关系已经属于 1NF，另外再满足一个条件，每个非主属性（不构成候选键的属性）都必须完全依赖于候选键，不能部分依赖于候选键，则称该关系属于第二范式（2NF）。即不能存在某个非主属性只依赖于候选键的一部分的情况。

通过拆分一个不属于 2NF 的关系为多个关系，可以使拆分后的关系属于 2NF。例如下述关系（其中用底划线标示的属性表示候选键）。

订单明细（订单编号，产品编号，产品名称，单价，数量）

订单明细关系的候选键是订单编号和产品编号的集合，非主属性产品名称和单价部分依赖于候选键，因为它也依赖于候选键的一部分（产品编号），所以这个关系不符合 2NF 的要求。

解决的办法是将订单明细关系拆分为两个关系，拆分后的两个关系都是属于 2NF 的（其中用斜体标示的属性表示外键）。

产品（产品编号，产品名称，单价）
订单明细（订单编号，*产品编号*，数量）

下面通过数据来说明这个问题，如表 2-14 所示的订单明细关系存在部分依赖。

表 2-14 存在部分依赖的“订单明细”关系

订单编号	产品编号	产品名称	单价	数量
1	1	U 盘（64G）	68	1
1	2	无线鼠标	56	2
1	3	无线路由器（4 口）	128	1
2	1	U 盘（64G）	68	2

2	3	无线路由器（4 口）	128	1

将“产品（产品编号，产品名称，单价）”关系拆分出来以后，消除了部分依赖，如表 2-15 所示，这时订单明细关系的产品编号（外键）参照产品关系的产品编号（主键）。

表 2-15 拆分后的“订单明细”关系（消除部分依赖）

“订单明细”关系

订单编号	产品编号（外键）	数量
1	1	1
1	2	2
1	3	1
2	1	2
2	3	1

“产品”关系

产品编号（主键）	产品名称	单价
1	U 盘（64G）	68
2	无线鼠标	56
3	无线路由器（4 口）	128

从表 2-15 可以看到，拆分订单明细关系的过程是将具有重复值的属性（产品编号、单价和产品名称）拆分出来，作为一个新的关系，并删除新关系中重复的行。原来表的产品编号作为外键，参照新关系的主键。拆分前存在着数据冗余（产品名称和单价两个属性），有可能出现更新异常、插入异常和删除异常，而拆分成两个关系则可以避免这些问题。

5. 第三范式（3NF）

如果一个关系已经属于 2NF，另外再满足一个条件，每个非主属性（不构成候选键的属性）都必须直接依赖于候选键，不能传递依赖于候选键，则称该关系属于第三范式（3NF）。即不能存在非主属性 A 依赖于非主属性 B，非主属性 B 再依赖于候选键的情况。

同样的，通过拆分一个不属于 3NF 的关系为多个关系，可以使拆分后的关系属于 3NF。例如下述关系。

```
订单（订单编号，客户编号，订单日期，客户姓名，客户地址）
```

在这个关系中，所有非主属性（订单日期，客户编号，客户姓名，客户地址）都完全依赖于候选键（订单编号），所以是 2NF 的。但是有两个非主属性（客户姓名，客户地址）通过客户编号传递依赖于候选键，所以不符合 3NF 的要求。

解决的办法是将订单关系拆分为两个关系，拆分后的两个关系都是属于 3NF 的。

```
客户（客户编号，客户姓名，客户地址）
订单（订单编号，订单日期，客户编号）
```

下面还是通过数据来说明这个问题，如表 2-16 所示的订单关系存在传递依赖。

表 2-16 存在传递依赖的“订单”关系

订单编号	客户编号	订单日期	客户姓名	客户地址
1	1	2024-08-12	练德生	江苏兴化市人民路 123 号
2	2	2024-08-12	刘健康	上海黄浦区侨汇路 216 号
3	2	2024-08-19	刘健康	上海黄浦区侨汇路 216 号
4	1	2024-09-02	练德生	江苏兴化市人民路 123 号

将“客户（客户编号，客户姓名，客户地址）”关系拆分出来以后，消除了传递依赖，如表 2-17 所示，这时订单关系的客户编号（外键）参照客户关系的客户编号（主键）。

表 2-17 拆分后的“订单”关系（消除传递依赖）

“订单”关系

订单编号	客户编号（外键）	订单日期
1	1	2024-08-12
2	2	2024-08-12
3	2	2024-08-19
4	1	2024-09-02

“客户”关系

客户编号（主键）	客户姓名	客户地址
1	练德生	江苏兴化市人民路 123 号
2	刘健康	上海黄浦区侨汇路 216 号

与前述订单明细关系的情况相似，从表 2-17 可以看到，拆分订单关系的过程是将具有重复值的属性（客户编号、客户姓名和客户地址）拆分出来，作为一个新的关系，并删除新关系中重复的行。原来的表的客户编号作为外键，参照新关系的主键。订单关系拆分前存在着数据冗余（客户姓名和客户地址两个属性），有可能出现更新异常、插入异常和删除异常，而拆分成两个关系则可以避免这些问题。

2.4.3 关系中异常的消除

对“2.4.1 关系中的异常”提出的例子进行分析，如表 2-8 所示的关系可表示为如下。

学生（班级名称，班主任，班主任电话，学号，姓名，性别）

这个关系中学号是主键，班主任和班主任电话通过班级名称传递依赖于学号，根据规范化设计的要求，拆分为如下两个关系，两个关系各添加一个主键，学生关系添加一个外键，参照班级关系的主键。

班级（id，班级名称，班主任姓名，班主任电话）
学生（id，学号，姓名，性别，***班级 ID***）

这时班级关系中，班主任电话通过班主任姓名传递依赖于班级编号，还要进一步拆分，结果如下。

班主任（id，班主任姓名，班主任电话）
班级（id，班级名称，***班主任 ID***）
学生（id，学号，姓名，性别，***班级 ID***）

拆分的过程也可以看成是将具有重复值的属性班级名称拆分出来作为一个新的关系，并将属于另一个实体的班主任和班主任电话拆分出来作为一个新的关系，为新关系添加主键，原来的关系添加外键，参照新关系的主键。

规范化后的关系中不存在部分依赖和传递依赖，因此符合 3NF 的要求。这时表 2-8 所示的学生关系经过拆分，成为如表 2-18 所示的三个关系，可以避免关系中异常的出现。

表 2-18 规范化后的“班主任”、“班级”、“学生”关系及数据

“班主任”关系

id	班主任	班主任电话
1	李进中	13587654321
2	汪一平	13712345678

“班级”关系

id	班级名称	外键（参照班主任 ID）
1	软件 31431	1
2	网络 31451	2

“学生”关系

id	学号	学生姓名	学生性别	外键（参照班级 ID）
1	3143101	张三	男	1
2	3143102	李四	女	1
3	3143103	王五	男	1
4	3145101	赵六	男	2

从表 2-18 可以看到，数据冗余已经达到最小，没有更新异常，例如修改班主任电话只需要修改一处，

没有删除异常，例如删除学生“赵六”时，不会同时删除所在的班级和班主任，没有插入异常，例如插入新老师“张明亮”时，只要在班主任关系中插入新老师，其他两个关系不受影响。

2.5 关系数据库设计 6 步实施法

学习关系数据库基础的目的不仅是要在关系模型的指导下，设计一个合格的关系数据库，还要在范式理论的指导下，设计出一个“好”的关系数据库。

2.5.1 “好”的关系数据库的要求

一个合格的关系数据库是要满足关系模型的三要素：（1）关系模型的数据结构，即关系的定义和 6 项基本特征；（2）关系模型的数据操作；（3）关系模型的数据完整性约束。

一个“好”的关系数据库是要在一个合格的关系数据库的基础上，再满足范式理论中的 1NF、2NF 和 3NF 的要求。

因此，需要满足下述三个要求。

- 满足关系模型的基本特征：主要是满足“表 2-4 关系的 6 项基本特征”。
- 满足关系模型的数据完整性约束：主要是主键约束和外键约束。
- 满足 1NF、2NF 和 3NF 的要求：1NF、2NF 和 3NF 的要求总结如表 2-19。

表 2-19 1NF、2NF 和 3NF 总结

范式	范式要求	解决办法
1NF	属性值应该是原子性的	两种办法：（1）拆分为多个属性；（2）拆分为多个关系
2NF	关系中不存在部分函数依赖	拆分为多个关系，达到表的原子性的要求
3NF	关系中不存在传递函数依赖	拆分为多个关系，达到表的原子性的要求

表 2-19 的范式要求理论性比较强，幸运的是，解决方法比较简单，就是将包含多个实体的表拆分为多个关系，使每个关系只包含一个实体，满足表的原子性的要求，就能达到 3NF 的要求。

因此，一个“好”的关系数据库要满足下述要求。

- 主键和外键：每张表必须有且有一个主键，通过外键建立表的联系。
- 属性的原子性：满足关系 6 项基本特征的要求。
- 属性值的原子性：满足 1NF 的要求。
- 表的原子性：满足 2NF 和 3NF 的要求。

2.5.2 规范化设计 6 步实施法

当深刻理解了关系数据库基础和关系数据库的设计知识后，就可以采用以下 6 步实施法进行规范化设计，将概念模型、关系模型和物理模型三个设计阶段合并起来同时进行。该 6 步实施法在采用 MySQL Workbench 建模工具设计数据库时也有所体现，详见单元 3“3.5 建模工具的使用”，读者可以对照学习。

1. 列出所有二维表

从需求分析中收集将要存入数据库的所有数据，列出所有二维表，不能有任何遗漏，也不要有重复。每张表不应只有列名，还应该包含测试数据，以便加深对数据之间联系的理解，更好的进行规范化。

2. 检查属性的原子性

按照关系的 6 项基本特征检查每一张表，要求每张表都要符合关系的 6 项基本特征，其中最有代表性的特征是属性的原子性，如果有不符合原子性的列，则需要对列重新命名，消除列的子属性。

完成后，这些表符合关系的基本特征，成为一个合格的关系模型。

3. 设置主键和外键参照

关系模型的数据完整性约束有下述两个基本要求。

- 主键约束：为每张表设置一个主键，通常是整型的，并且设置为自动增量。
- 外键约束：通常每张表至少与另一张表有联系，要么参照别的表，要么被别的表参照，也可能两者兼而有之，只在极少情况下会出现独立的表。

完成后，可以满足关系模型的数据完整性约束的要求。

4. 检查属性值的原子性

检查所有表，找出属性值中包含多个值的属性（某一行的属性值中包含多个值），例如前述包含两个电话号码的属性（参见表 2-11）。可以根据业务需求采用下述方式中的一种进行处理。

- 拆分属性：将属性拆分为多个属性，分别保存多个值。这种方式的缺点是只能保存有限个值。
- 拆分表：将属性独立出来成为一张表，为新表添加主键和外键，这个外键参照原表的主键。

完成后，可以达到第一范式（1NF）的要求。

5. 检查表的原子性

检查所有表的原子性，即检查表是否包含多个实体，如果包含多个实体，则需要拆分表，使每张表只包含一个实体，达到表的原子性的要求。

一张表包含两个实体有两种表现形式，一是表内两个实体是一对多联系，二是表内两个实体是一对一联系。下面分别讲解。

1) 表内实体是一对多联系

表内实体是 1:n 联系时，将会出现含有重复值的属性，具有重复值的属性常常是属于另外的实体，表示这张表包含两个实体，不满足表的原子性的要求，这时常常要对表进行拆分。根据具体情况，采用如表 2-20 所示的方式中的一种进行处理。

表 2-20 处理属性值重复的几种方式

类型	处理方式
简单的值	不需要拆分，还是作为属性。例如“性别”属性的值只有“男”和“女”二个，属于简单的值，不需要拆分，也可以采用下述内部编码方法进行处理。
内部编码	如果重复值的数量是有限和较少的，并且是固定不变的，这时可以采用内部编码来替代重复的值。例如“性别”属性的值只有“男”和“女”二个，这时采用内部编码 M 替代“男”，用 F 替代“女”。又如用户的“状态”属性只有三种：“待激活”、“激活”和“禁用”时，可以用内部编码 0、1、2 分别代表“待激活”、“激活”和“禁用”。
拆分表	将含有重复值的属性独立出来成为一张新表，为新表添加主键，并删除重复值，同时将与该属性有关的其他属性也并入新表。原表和新表之间是多对一的联系，原表添加外键，参照新表的主键。

但是要注意下述两种情况。

- 假性重复：例如成绩列会有许多相同的值，但不能认为是重复值，因为相同的成绩在本质上不是重复，只是碰巧出现了相同的值。
- 隐性重复：没有在测试数据中反映出来的重复，当数据量足够大时，某些属性的值可能会出现重复，这种情况也应该加以考虑。

2) 表内实体是一对一联系

另外还要考虑一种特殊的情况，一张表包含了两个实体，但是并没有出现属性值重复的现象，这是因为这两个实体是 1:1 的联系（如果是隐性重复，也有可能是前述的 1:n 的联系），这时也应该将这两个实体拆分开来，两个实体分别设置主键和外键，从表的外键参照主表的主键，从表的外键添加唯一性约束。

完成后，所有表都是原子性的，不会出现部分依赖和传递依赖，可以达到 2NF 和 3NF 的要求。

6. 合并相同的实体

前述步骤拆分出来的实体可能存在相同的实体，相同的实体一般具有相同的主键和相同的属性，属性可能全部相同，也可能部分相同，应该将相同的实体合并成一个实体。

至此，规范化设计全部完成。

前述 6 个步骤总结如表 2-21 所示，最重要的原则是“三个原子性”，即属性的原子性、属性值的原子性和表的原子性。

表 2-21 规范化设计的 6 步实施法

步骤	说明
1、列出所有二维表	收集和整理所有数据，以二维表的形式列出，不要遗漏，不要重复，并填入测试数据
2、检查属性的原子性	每张表都要符合关系的 6 项基本特征，特别是属性的原子性，解决方案是对属性重新命名
3、设置主键和外键	为每张表设置主键，找出表之间的联系（一对一、一对多、多对多），为从表设置外键，参照主表的主键，一对一的外键添加唯一性约束。对于多对多联系，还要将联系转换为新的表，新表的两个外键分别参照原来两张表的主键
4、检查属性值的原子性	检查所有列的值的原子性，如果不是原子的，则（1）拆分属性；（2）拆分表
5、检查表的原子性	检查所有列的值的重复性，如果不是假性重复，则（1）内部编码；（2）拆分表 找出包含两个实体的表（1:1 或 1:n 联系两种情况），并进行拆分
6、合并相同的实体	检查所有表，合并相同的表

2.5.3 规范化设计中的一些问题

1. 主键如何确定

每张表必须有且只有一个主键，主键是候选键中任意指定的一个，任何一个候选键都可以作为主键使用，例如学生的学号、身份证号都可以作为主键使用，图书的 ISBN 书号也可以作为主键使用。

目前标准的做法是使用无任何业务含义的整数作为主键，这样做的好处是，主键值可以自动生成，主键值也无需修改。

2. 外键是否一定要参照主表的主键

外键不是必须参照主表的主键。设置外键的原则是，外键只能参照主表上有唯一性约束的列，就是说，可以参照主表的任何一个候选键，而不能参照候选键之外的列。

例如一张表的外键（身份证号）参照另一张表的候选键（身份证号），但是尽量不要这样做。

3. 需求分析要做到什么程度

需求分析在项目开发中的重要性是无容置疑的，需求分析可以做得非常深入，也可以说是无穷无尽的，究竟要做到什么程度是需要结合具体的需求来确定的，并不是做得越深入越好。

一般来说，小型项目的需求分析要求较低，因此规范化设计也比较简单，项目越大，需求分析的要求越高，规范化设计也越严格。只要把用户的需求全部考虑周全，能够避免开发过程中和项目验收时提出新的需求或变更需求，需求分析就基本到位了。有时还会预留一些功能，以备不时之需，例如开发销售系统时，用户没有提出打折销售的需求，而设计时可以预留这项功能，因为这是所属领域的常见需求。如果在设计时能够预见到一些需求，而提前做好应对措施，这样的项目就更容易成功。

需要注意的是，针对同一个实体，不同的需求可能会有不同的分析结果。例如对于图书实体，如果项目的需求是图书的销售管理，那么图书实体的属性主要是书号、作者、书名、价格、出版年份等，如果项目的需求是图书的印刷管理，那么图书实体的属性主要是书号、开本、纸质、装订方式、印色、页数、印

刷数量、交货日期等。

4. 规范化设计要做到什么程度

规范化设计也不是越彻底越好，总的原则是要达到 3NF 的要求。在实际开发中，可能还会有一些实际的考量，分为下述两种情况加以考虑。

- 小型项目的规范化设计比较简单，在某些局部会允许不满足 1NF 或 2NF 或 3NF 要求的情况出现，但是大型项目的规范化设计就会比较严格，甚至要达到更高的范式要求，例如 BCNF。
- 对于数据量大的表，例如行数达到几千万的表，出于性能的考虑，需要一定的数据冗余，在事关性能的关键设计上，故意不满足规范化设计的要求，这种情况也称为反规范化设计，这是一种用空间换时间的折衷方案，由于数据冗余，占用较大的存储空间，换来了少一个内连接，提高了性能。

【案例讲解】书店管理项目的设计

任务 1 需求分析

需求分析是项目开发中最重要的一步，要与用户进行深入的讨论，充分理解项目的目的和要求，决定要实现哪些功能，不需要哪些功能，根据需求分析的结果写成需求分析报告，签订开发合同。当项目结束时，将根据开发合同和需求分析报告来评估项目是否完成。

5. 需求描述

1、 项目概况

项目名称：书店管理项目 BookStore（缩写为 bs）。

目标用户：线下的小型书店。

2、 需求概述

本项目是一个线下小型书店管理系统，用于管理图书的销售，查询和统计销售的情况。

6. 需求调查

在真实的项目中，需求调查是最费时的一个阶段，调查的内容要非常详尽，例如开销售单时，是否允许开单的销售人员修改价格，如果允许修改，那么折扣是如何确定的，如果折扣超过一定的限度，是否需要审批。又如销售的处理流程，开好销售单后，下一步是由哪个部门处理，是否需要审批，审批的权限如何确定，收款是在哪个环节完成的，是否允许赊账，是否需要预付定金，定金的比例是如何确定的，发货是由哪个部门处理，是客户自提，还是通过第三方物流公司送货，是否有分批发货的现象，如果缺货又是如何处理的。

3、 调查内容

需求分析的第一步是调查用户的实际需求，调查的主要内容如下所示。

- 调查组织机构：了解用户的部门组成情况、各部门的职责，重点是与项目有关的情况。
- 调查业务流程：调查用户的业务活动，收集各个部门输入和使用的数据，充分理解数据的用途和作用、对数据的处理加工过程、数据的格式要求等，还要理解数据在各个部门之间流通交换的过程，部门之间是如何衔接的。
- 确定数据规范：与用户讨论确定对数据的规范化要求，如处理要求、格式要求、完整性要求和安全性要求等，例如商品的计量单位是什么，如果是称重的，称重的精度是多少。
- 确定系统边界：与用户讨论确定项目应该实现的功能，项目的功能不是无限的，必须在需求调查阶段确定要实现哪些功能，不需要哪些功能，以避免开发过程中不断增加新的功能而导致项目的返工或重建。

4、 调查办法

需求调查的方法根据不同的问题和条件，会有不同的办法，主要的办法如下所示。

- 开会调查：与用户的相关人员召开座谈会，调查用户的需求和业务流程，会议应该是讨论式的，而不是讲课式，会上要有充分的交流。
- 随岗调查：对于重要的业务，可以直接参加到业务中去，与员工一起上岗工作，亲身体验业务的开展过程，直接获得第一手资料。
- 问询调查：对于重要的岗位和人员，进行个别的沟通和交流，特别要重视关键岗位的业务流程和业务需求。
- 问卷调查：将调查问卷发放到相关岗位的员工，收集数据。这种办法要求调查问卷设计得合理，但这并不容易做到，因此并不常用。
- 档案调查：查阅用户的档案资料，收集相关的数据。这种办法非常有效，但是许多用户出于各种原因，可能不太愿意配合。

7. 数据收集

在调查的过程中，要特别重视数据的收集，这些数据包括但不限于如下所示。

- 各种单据：各种纸质和电子的单据，如销售单、发货单、销售小票、账务收据等。如果已经电子化了，则收集电子格式或打印的单据。
- 统计报表：各种统计、汇总、上报、下达的报表，也包括 Excel 制作的报表。
- 业务分析：各种工作记录、业务流程、会议记录、业务分析报告等，只要与项目有关的都要收集。

收集的数据中应该包括数据的用途、作用、格式、数据的流向、数据在各部门流转过程中的状态，还要包括一些涵盖了各种情况的例子数据，以便更好地理解数据。还要注意一些例外情况的处理。

任务 2 系统功能分析

在前述几个步骤的调研后，经过开发人员与用户的讨论分析，并经用户确认，最后确定项目的功能。例如本项目用户对系统功能需求如图 2-19 所示，这些功能体现在如图 2-20 所示的左侧菜单中。

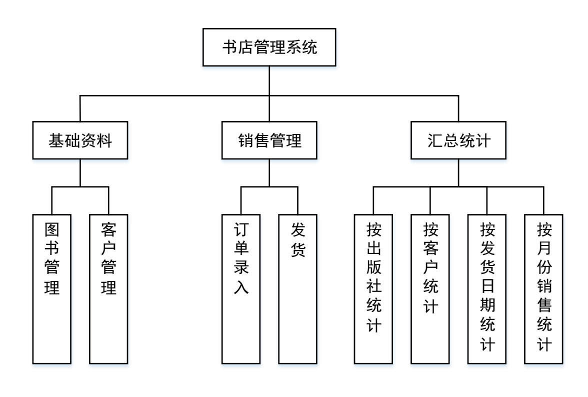


图 2-19 系统功能需求



为方便读者理解“书店管理”项目，本小节将用户的需求以一个完成了的、可运行的项目来展现，如图 2-20 所示仅仅是其中一个界面。这些界面展现了项目所管理的数据，以及项目的业务处理流程。读者可以在浏览器中打开附录 D 的“项目 2c 书店管理项目”，在项目的运行过程中体验用户的需求和系统的功能，读者在体验项目的功能后，再继续进行规范化设计。



图 2-20 书店管理项目的图书管理

任务 3 规范化设计

再对以图 2-20 为代表的各个功能界面进行分析，图书管理功能由出版社和图书两个实体组成，客户管理功能只有一个客户实体，订单录入功能由订单和订单明细两个实体组成。

因此，最终确定共有 5 个实体：出版社、图书、客户、订单和订单明细，这五个实体的 ER 图参见“2.3.2 ER 模型”一节的图 2-9，其中订单明细以多对多的联系出现。

将上述 5 个实体转换为关系（表），得到如表 2-22～表 2-26 所示的 5 张表的结构，这是按照规范化设计的要求，设计出来的关系数据库的数据结构。

表 2-22 是出版社表，它不参照其他表，但它被图书表参照。

表 2-22 bs_publisher（书店管理_出版社）

序号	列名	显示标题	数据类型	为空性	唯一性	默认值	备注
1	id	ID	int	not null	uqinue		主键，自增量
2	col_name	出版社	varchar(50)	not null			
3	col_addre	地址	varchar(50)	not null			
4	col_tel	联系电话	varchar(50)	not null			

表 2-23 是图书表，它参照出版社表，同时它也被订单明细表参照。

表 2-23 bs_book（书店管理_图书）

序号	列名	显示标题	数据类型	为空性	唯一性	默认值	备注
1	id	ID	int	not null	uqinue		主键，自增量
2	col_author	作者	varchar(50)	not null			
3	col_title	书名	varchar(50)	not null			
4	col_year	出版年份	year	not null		2155	
5	col_isbn	ISBN 书号	varchar(50)	not null			
6	col_classification	分类号	varchar(50)	not null			
7	col_price	价格	decimal(13,2)	not null		0.00	
8	id_bs_publisher	出版社 ID	int	not null		0	参照 bs_publisher 表

有些读者会认为一本书的作者可能有多个，因此表 2-23 设计的“作者”列不满足 1NF 的要求，即属性值不是原子性的。这需要根据用户的需求来确定，对于图书销售，作者列只需要罗列所有作者即可，如

果是出版社内部的管理，就需要有作者表，包含作者的工作单位、手机、联系地址和个人简介等信息，作者表与图书表是多对多的联系，还要有图书作者表，它参照图书表和作者表，还包含了作者的排名，是主编还是副主编等信息。

表 2-24 是客户表，它被订单表参照。

表 2-24 bs_customer（书店管理_客户）

序号	列名	显示标题	数据类型	为空性	唯一性	默认值	备注
1	id	ID	int	not null	uqinue		主键，自增量
2	col_name	姓名	varchar(50)	not null			
3	col_sex	性别	char(1)	not null			
4	col_tel	电话	varchar(50)	not null			
5	col_addre	默认送货地址	varchar(50)	not null			

表 2-25 是订单表，它参照客户表。注意一个完整的订单应该由两部分组成，一是订单头表，也就是这里的订单表，二是订单行表，也就是如表 2-26 所示的订单明细表。

表 2-25 bs_order（书店管理_订单）

序号	列名	显示标题	数据类型	为空性	唯一性	默认值	备注
1	id	ID	int	not null	uqinue		主键，自增量
2	id_bs_customer	客户 ID	int	not null		0	参照 bs_customer 表
3	col_addre	实际送货地址	varchar(50)	not null			
4	col_total_amount	总金额	decimal(13,2)	not null		0.00	
5	col_status	状态	varchar(50)	not null			
6	col_order_date	订单日期	datetime	not null		2155-12-31 23:59:58	
7	col_shipping_date	出库日期	datetime	not null		1901-01-01 00:00:00	
8	col_remark	备注	text	null			

表 2-26 是订单明细表，也称为订单行表，它记录了一个订单中的详细信息，它参照图书表和订单表。

表 2-26 bs_order_detail（书店管理_订单明细）

序号	列名	显示标题	数据类型	为空性	唯一性	默认值	备注
1	id	ID	int	not null	uqinue		主键，自增量
2	id_bs_order	订单 ID	int	not null		0	参照 bs_order 表
3	id_bs_book	图书 ID	int	not null		0	参照 bs_book 表
4	col_price	单价	decimal(13,2)	not null		0.00	
5	col_quantity	数量	int	not null		0	
6	col_amount	金额	decimal(13,2)	not null		0.00	

至此，书店管理项目的数据结构设计完成。

【实战演练】图书借阅项目的设计

请读者根据下述需求自行对图书借阅项目进行规范化设计，得到一个关系数据库的数据结构。

任务 1 需求分析

8. 项目概况

项目名称：图书借阅项目

目标用户：适用于小型单位（50~200 人），管理数千本图书的借阅

9. 需求概述

本项目是一个小型图书借阅系统，用于管理收藏的图书，记录借阅情况，查询图书状态。

任务 2 系统功能分析



本小节以完成后的项目界面来说明用户的需求。读者可以在浏览器中打开“项目 2d 图书借阅项目”，在项目的运行过程中体验用户的需求和分析系统的功能，功能包括基础资料的管理、借还书管理、以及查询统计等，如图 2-21 所示。



图 2-21 图书借阅项目的图书及副本管理

任务 3 规范化设计

根据前述【案例讲解】中对需求分析的要求，参考书店管理项目的设计，对图书借阅项目进行需求分析，自行完成“图书借阅”数据库的规范化设计，要求如下。

1) 设计要求

在充分理解关系数据库的基础上，按照规范化设计的要求，对图书借阅项目进行规范化设计。

2) 命名要求

数据库名：library。

表名：以 lib_作为前缀，即 Library 的前 3 个字母。

列的命名：

- 主键列的名称统一为“id”。
- 外键列的名称为“id_被参照的表名”，例如 id_lib_book。
- 其他列的名称以 col_为前缀，后接该列的英文名称，不要用拼音或拼音首字母。

3) 格式要求

规范化设计的成果以表 2-27 所示的形式展现，以该格式列出每张表的数据结构设计。

表 2-27 lib_xxxx（图书借阅_XXXX）

序号	列名	显示标题	数据类型	为空性	唯一性	默认值	备注
1	id	ID	int	not null	uqinue		主键，自增量
2	col_xxx	XXX	varchar(50)	not null			

【单元重点】

单元小结参见本单元的【思维导图】，单元重点如下。

- 数据模型：三要素（数据结构、数据操作、数据完整性约束），三层次（概念、逻辑、物理）。
- ER 模型：常用术语（实体集、实体、属性、属性值、域、键）、ER 图、实体和实体间的三种联系（一对一，一对多，多对多）。
- 关系模型：术语（关系、元组、属性，以及表、行、列），关系的 6 项基本特征，关系的表示，候选键、主键、外键和非主属性，主表和从表。
- ER 模型向关系模型的转换，三种联系（一对一，一对多，多对多）的转换。
- 主键：用于唯一标识表中的每一行。每张表必须有一个，且只能有一个主键，主键必须唯一、不能为空。
- 外键：用于参照主表的某一行，建立表之间的联系。每张表可以有一个或多个外键，也可以没有外键，但它的值必须是所参照的主表的主键值中的一个。
- 关系数据库设计："好"的关系模型要求有尽量低的数据冗余、没有插入异常、更新异常、和删除异常，因此要满足 1NF（属性值的原子性）、2NF（没有部分依赖）、3NF（没有传递依赖）的要求。
- 设计关系数据库：三个原子性（属性的原子性、属性值的原子性、表的原子性），尽可能拆分（拆分属性、拆分属性值、拆分表），从而消除关系中的异常。

【课后思考】

一、选择题

6. 数据模型的三要素是【　　　　】。

   A. 数据结构　　B. 数据操作　　C. 数据控制　　D. 数据完整性约束

7. ER 模型是（　　）。

   A. 一种概念模型　　B. 一种逻辑模型　　C. 一种物理模型　　D. 一种数据库模型

8. 关系模型是（　　）。

   A. 一种概念模型　　B. 一种逻辑模型　　C. 一种物理模型　　D. 一种数据库模型

9. ER 模型可以表示（　　）。

   A. 一对一联系　　B. 一对多联系　　C. 多对多联系　　D. 上述三种联系

10. 关系模型可以表示（　　）。

    A. 一对一联系　　B. 一对多联系　　C. 多对多联系　　D. 上述三种联系

11. 实体间的多对多联系可以转换为（　　）。

    A. 两个一对一联系　　B. 两个一对多联系　　C. 两个多对多联系　　D. 以上都不是

12. 以下哪项不是关系模型的基本特征（　　）。

    A. 行的唯一性　　B. 列的唯一性　　C. 属性的原子性　　D. 属性值的原子性

13. 下述关于主表和从表的说法，正确的是【　　　　】。

    A. 主表是被参照的表　B. 从表是被参照的表　C.主表必定拥有外键　D. 从表必定拥有外键

14. 数据冗余可能导致【　　　　】。

    A. 插入异常　　B. 删除异常　　C. 更新异常　　D. 查询异常

15. 一个"好"的关系模型需要满足哪些要求（　　）。

    A. 满足关系模型的基本特征　　B. 满足关系模型的数据完整性约束

C. 满足 1NF、2NF 和 3NF 的要求　　　　D. 以上都是

二、填空题

16. 主键的作用是____________________________，它的值必须________，并且________。

17. 外键的作用是_______________________，它的值必须___________________________。

三、思考题

18. ER 模型和关系模型有什么关系？

19. 如何将 ER 模型转换为关系模型？

20. 一个“好”的关系模型的要求是什么？如何设计一个“好”的关系数据库？

【课外拓展】

5、 问一问 AI，关于 ER 模型、局部 ER 图、全局 ER 图，以及 ER 模型向关系模型转换的知识。

6、 问一问 AI 有关范式理论的问题，加深对范式理论的理解。

7、 问一问 AI，数据库的三级模式（外模式、概念模式和内模式）、数据库的二级映射（外模式/模式映射、模式/内模式映射）的知识。

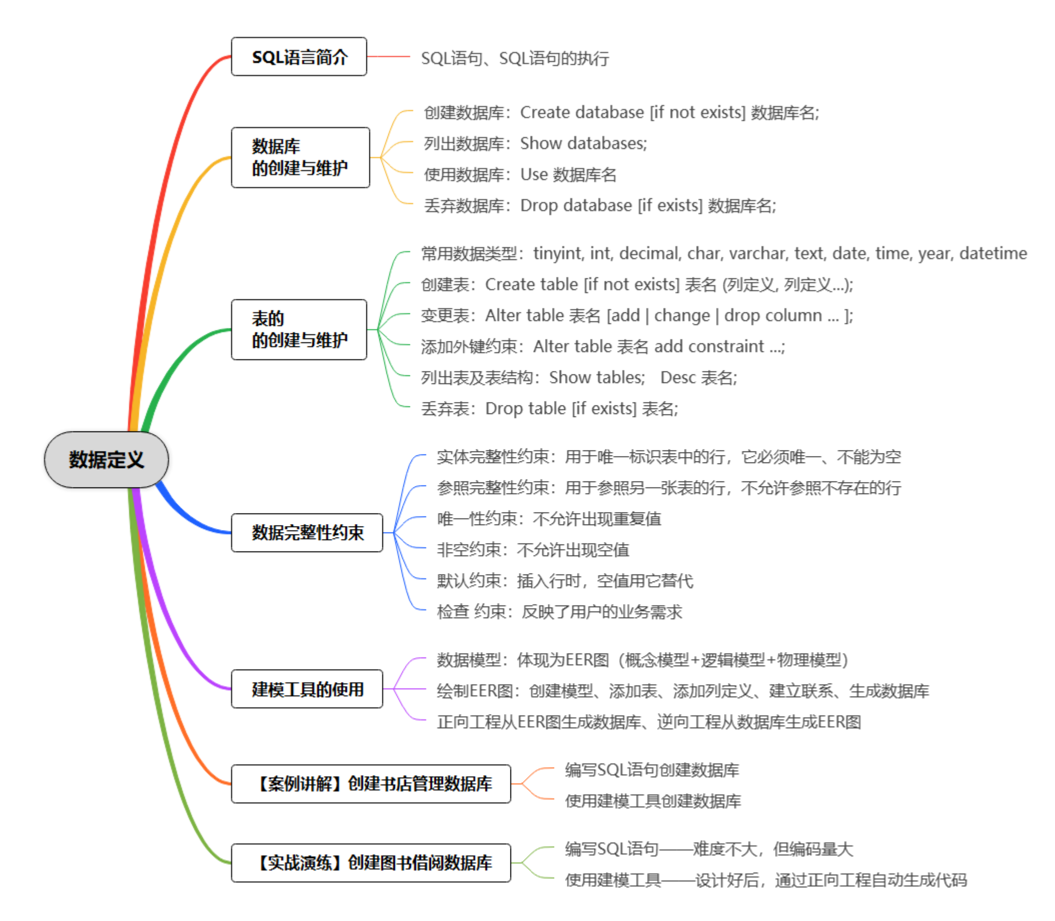
8、 从网上查找名为“阿里巴巴 Java 开发手册-2022.pdf”的文件，阅读其中关于表设计的部分。

单元3 数据定义

【学习目标】

知识目标	能力目标
◆ 了解字符集和校对。	◆ 学会编写和执行 SQL 语句。
◆ 理解常用的数据类型。	◆ 掌握数据库的创建与维护。
◆ 深刻理解一对一、一对多和多对多联系。	◆ 掌握表的创建、变更与维护，外键的添加。
◆ 深刻理解 6 种数据完整性约束。	◆ 初步学会使用建模工具。
◆ 理解 EER 图的含义与作用。	**素质目标**
◆ 理解建模工具与数据库设计的关系。	◆ 培养良好的编程习惯，遵守命名规范
◆ 初步理解正向工程和逆向工程。	◆ 理解规则的重要性，培养遵守法律法规

【思维导图】



【情景导入】

小明学习单元 2 时，一开始觉得特别难，有许多术语和概念，后来发现所有理论都是为了设计一个好的关系数据库，核心概念是一对一、一对多和多对多，这 3 种联系通过外键参照主键实现。在掌握了关系数据库 6 步实施法，从属性的原子性、属性值的原子性和表的原子性来思考问题，就觉得容易多了。听学长说，这是数据库课程最难的部分，因此，小明认为难的都已经学完，就更有信心学好这门课，让我们

与小明一起继续学习吧。

【知识储备】

单元 2 讲解了关系数据基础知识，以及如何设计一个好的关系数据库。本单元将要根据单元 2 的设计成果，在 MySQL 上加以实施，创建数据库和创建表。

单元 1 和单元 2 分别体验过创建数据库、创建表、输入数据、查询数据的开发过程，都是在 MySQL Workbench 中用图形界面完成的。从本单元开始，将要学习使用 SQL 语句来完成这个过程，像个真正的程序员一样，编写一行行代码，完成开发任务。

3.1 SQL 语言简介

SQL 语言是学习关系数据库管理系统的基础，MySQL、Oracle、SQL Server 和 SQLite 等所有关系数据库管理系统用的都是 SQL 语言。学好 SQL 语言是本书的目标，从现在开始，直到全书结束，基本上不再使用图形界面，而是使用 SQL 语言实现所有功能。



SQL 是一种数据库编程语言，它与 C++、Java 或 Python 等通用计算机语言不同，后者用于编写各种各样的程序，而 SQL 是设计用来管理关系数据库的。

3.1.1 SQL 语句

SQL 语言的代码由 SQL 语句组成，它的基本执行单元是 SQL 语句，可以直接执行一条 SQL 语句。而在 C++、Java 或 Python 等其他编辑语言中，基本的执行单元是函数，不允许直接执行一条语句，语句必须放在函数中才能被执行。

一条 SQL 语句由一个 SQL 命令关键字开始，后接该语句要求的关键字和标识符。

例如单元 1“Hello，数据库”项目中，使用过下述语句。

```
Select id 序号, author 作者, title 书名, price 价格, publisher 出版社
    from book_list;
```

这是一条 select 语句，它的命令关键字是 select，它还用了一个 from 关键字，其余部分是标识符。

注意以下几点。

- SQL 命令关键字和关键字是大小写无关的，例如 Select、SELECT 和 select 都是相同的。
- SQL 标识符从理论上讲是大小写无关的，例如 author 和 Author 是相同的，但是在以下情形下，仍然是大小写有关的：（1）在 Linux 操作系统中，数据库名和表名等是大小写敏感的；（2）当标识符用于标题显示时，也是大小写敏感的；因此要养成区分大小写的良好习惯，区分大小写还有助于对 SQL 代码的理解，避免因大小写不同而引起的混淆。
- SQL 标识符从原则上讲是不允许使用命令关键字和关键字的，如果由于某种原因，一定要使用关键字作为标识符，则需要用反引号（位于键盘 TAB 键的上方）括起来，例如`Select`，一旦这样用了，以后引用这个标识符时，也要加上反引号，引起诸多不便，因此不建议这样命名。
- 一条 SQL 语句应该以分号结束，在没有歧义时，语句末尾可以没有分号，但是在语句末尾加上分号是一个良好的习惯。在有些情况下必须在语句末尾加上分号，例如在单元 7 中讲解数据库编程时，就必须在语句末尾加上分号，否则出错。

本书将要学习的命令关键字如表 3-1 所示，学好这 9 个命令关键字，基本上就学好了 SQL。

表 3-1 SQL 常用命令关键字

分类	命令关键字	说明	相关章节
数据定义语言（DDL）	Create, Alter, Drop	数据结构的创建、变更和丢弃（删除）	单元 3
数据操纵语言（DML）	Insert, Update, Delete	数据的插入、更新和删除	单元 4
数据查询语言（DQL）	Select	数据的查询	单元 5
数据控制语言（DCL）	Grant, Revoke	权限的授予、撤回	单元 8

如同其他编程语言，SQL 代码也可以加上注释，单行注释以两个减号加一个空格开始，两个减号及其之后的内容是注释。多行注释是以/*开始，以*/结束。例如下述代码含有两种注释。

```
/*
多行注释：这是一条查询语句
SQL 语句有点像英语中的祈使句，大致的结构是：动词 + 宾语 + 介词
*/
Select id 序号, author 作者, title 书名, price 价格, publisher 出版社    -- 以命令关键字 Select 开始
    from book_list;                                                  -- 以分号结束一条语句
```

3.1.2 SQL 语句的执行

SQL 语句不需要编译就可以执行，SQL 语句通常保存在 SQL 文件中，因此，在 MySQL Workbench 单击左上角的“SQL”工具图标（参见单元 1 的图 1-28），创建一个 SQL File，然后在这个文件内编写 SQL 语句，单击“保存”图标就能保存这个文件。

执行时是以交互式的方式执行的，通常是编写一条语句，然后执行这条语句（单击“执行”图标，黄色的象“闪电”的图标），再编写下一条语句，再执行。SQL 文件并不是一定要保存，只在需要保留备用时才会保存 SQL 文件。

有如下两种执行方式。

- 执行 SQL 文件中的所有语句：直接单击“执行”图标，这时会按语句在文件中的先后次序执行 SQL 文件中的每一条语句，在单元 1 和单元 2 的例子中都是这样做的。
- 执行 SQL 文件中的部分内容：一个 SQL 文件中可能有多条 SQL 语句，这时可以选择性地执行其中一条或多条语句，甚至还可以选择性地执行一条语句中的部分内容，办法是用鼠标将想要执行的部分选择加亮，单击“执行”图标，这时只执行选中加亮部分的语句，如果选中加亮的部分不能构成完整的语句，则会出错。

3.2 数据库的创建与维护

与项目有关的数据、代码都是以数据库对象的形式保存在数据库中的，每个项目都有一个数据库，在数据库中保存项目的数据和代码。

创建与维护数据库的 SQL 命令关键字是 Create, Alter, Drop，它们分别用来创建、变更和丢弃数据库，在开始之前，先讲解字符集和校对，这对处理中文是十分重要的。

3.2.1 字符集和校对

1. 字符集

字符集（Character Set，缩写为 Charset）是字符编码中的所有字符，指定字符集的同时也就指定了字符编码。MySQL 支持几十种字符集，与中文有关的常用字符集是 GBK 和 UTF，字符集 UTF 还细分为多种：UTF8、UTF16、UTF32 和 utf8mb4 等，表 3-2 说明了几种常用字符集的区别。如果字符集设置错误，可能会出现中文乱码的现象。

表 3-2 几种常用字符集的区别

字符集	特点	容纳的字符	用途
GBK	固定占用两个字节	以常用汉字为主	Windows 操作系统使用
UTF8	最多占用三个字节	全球多数文字，包括中、日、韩三国的大多数汉字	MySQL 的早期版本使用
utf8mb4	最多占用四个字节	全球所有文字，以及表情字符，还包括了极冷僻的汉字	MySQL 8 的默认字符集

字符集 utf8mb4 是在 MySQL 5.5.3 开始引入的，MySQL 8.0 建议在任何情况下都要使用 utf8mb4，因而也是它的默认字符集。

2. 校对

校对（Collation）是指定字符的排序规则，例如英文的排序有两种规则：一种是区分大小写的排序，另一种是不区分大小写的排序。

对于 utf8mb4 字符集，常用的校对有下述两种。

- utf8mb4_unicode_ci：基于标准 Unicode 的排序，能够在各种语言之间精确排序。
- utf8mb4_general_ci：没有实现 Unicode 排序规则，在遇到某些特殊语言或者字符集时，排序结果可能不一致，但是速度较快。

一般情况下，采用默认的校对即可。如果要实现按汉语拼音进行排序，需要将其从 utf8mb4 字符集转换为 gbk 字符集之后再排序。

3. 使用字符集和校对

MySQL 支持在下述 4 个级别上设置字符集和校对。

- 服务器级字符集和校对：需要在配置文件中指定，MySQL 8.0 的默认字符集为 utf8mb4。
- 数据库级字符集和校对：在创建数据库时指定，默认采用服务器级字符集配置。
- 表级字符集和校对：在表级也可以指定，默认采用数据库级的字符集和校对。
- 列级字符集和校对：在列级也可以指定，默认采用表级的字符集和校对。

为了获得最大的兼容性，应该采用 utf8mb4 字符集，安装时服务器级的默认字符集是 utf8mb4，因此通常无需在更低的三个级别上设置字符集。

3.2.2 创建数据库

创建数据库可以采用两种方式，一种方式是采用图形界面的管理工具来创建数据库，在单元 1 和单元 2 的例子中已经体验过，另一种方式是采用 SQL 语言来创建数据库。

创建数据库的 SQL 语句的语法格式如下所示。

```
Create {database | schema} [if not exists] 数据库名
        [default character set 字符集 | collate 校对];
```

在语法格式中，“{ | }”表示二选一，“[ ]”表示可选项。参数说明如下。

- 数据库名：将要创建的数据库的名字，在 MySQL 中，数据库名必须唯一。
- database | schema：必须二者取一。在单元 1 讲解 MySQL Workbench 的界面时，讲过在 MySQL 中，database 与 schema 是等价的，可以互相替换，而在其他 DBMS 中，创建数据库时只能用 database，因此建议使用 database 关键字，以兼容其他 DBMS。
- if not exists：可选的。省略时，如果所指定的数据库已经存在，重复创建就会提示出错。加上这个选项则可以避免这种出错提示。
- 字符集：指定字符集。如果没有这个选项，则采用默认的 utf8mb4 字符集，建议采用默认值。
- 校对：指定校对。如果没有这个选项，则采用当前字符集的默认校对，建议采用默认值。

例如下述语句创建一个名为 abc 的数据库。

```
Create schema abc;
```

或者

```
Create database abc;
```

执行上述第二条语句的操作过程如图 3-1 所示。

1) 单击左上角的“SQL”工具图标，打开一个 SQL File 选项卡
2) 在 SQL 文件中编写 SQL 语句，如图 3-1 所示的文件中有两条语句，其中有两个汉字没有加上注释标记，MySQL Workbench 用红色的标记来提示出错。
3) 这两条语句是等价的，只需要执行其中一条即可，图中加亮选择了第二条语句。

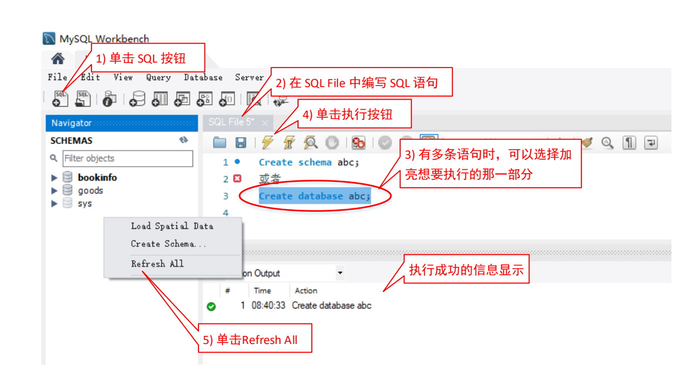


图 3-1 用 SQL 语句创建数据库 abc

4) 单击执行按钮，这时执行加亮的第二条语句，在下方的信息区显示了这条成功执行的语句。
5) 成功执行后，左侧的导航栏并没有列出新建的数据库 abc，这时要通过导航栏右键菜单中的 Refresh all 刷新一下，才能看到新建的数据库名称。

通过编写 SQL 语句来创建数据库与单元 1 的例子中通过图形界面创建数据库（参见单元 1 图 1-19）的效果是等价的，如单元 1 图 1-20 所示，图中显示执行的是 Create schema …;语句。

3.2.3 列出数据库

在左侧的导航栏中会列出所有数据库名，但是用下述语句也可以列出所有数据库的列表。

```
Show databases;
```



注意：databases 是复数形式，单词末尾加了表示复数的 s，区分单数和复数在 SQL 语句中十分常见，需要加以注意。

执行的结果如图 3-2 所示。

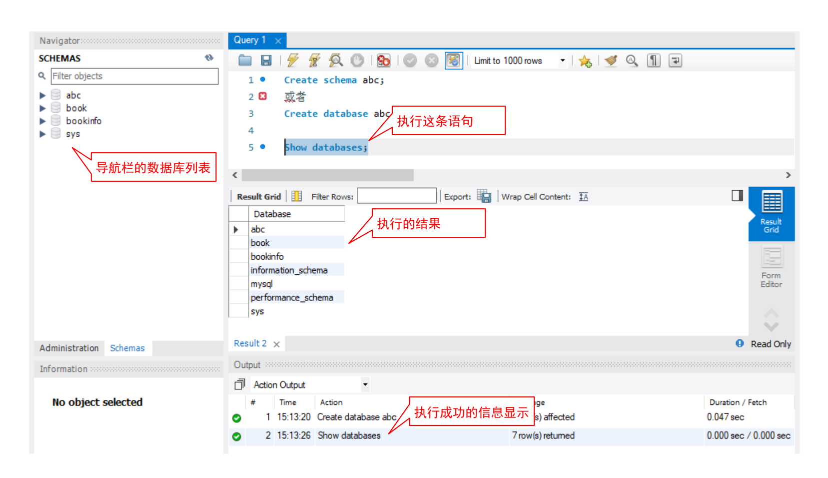


图 3-2 列出所有数据库

列表中的 book 数据库是单元 1 创建，bookinfo 数据库是单元 2 创建的，abc 数据库是刚才创建的。

对比导航栏中的数据库列表与 Show 命令执行结果的数据库列表，后者多出 3 个数据库，这 3 个数据库与 sys 数据库共同组成了 MySQL 的 4 个内置数据库，这 4 个内置数据库的作用如表 3-3 所示。

表 3-3 MySQL 的内置数据库

内置数据库	说明
mysql	MySQL 的核心系统数据库，它保存了用户账户信息、权限信息、系统变量、插件信息等内容。例如执行 show databases;时，实际上就是查询这个数据库中的信息
information_schema	从这个数据库可以查询关于所有数据库、表、列、索引、权限等的详细信息，实际的数据仍然是保存在 mysql 中
performance_schema	用于收集 MySQL 服务器性能诊断信息，通过分析这些数据，可以优化查询性能、监控资源使用情况等
sys	这个数据库提供了日常运维工作所需的一些工具，这些工具是一些视图、存储函数和存储过程，这些工具从 performance_schema 和 information_schema 中提取数据，提供了更加直观和易于理解的性能监控和诊断信息



对于初学者，并不需要关注这几个内置数据库，但是千万不能删除它们。

3.2.4 使用数据库

在 MySQL 中，数据库是一个存储数据库对象的容器，数据库对象是指表（Table）、视图（View）、存储过程（Stored Procedure）和存储函数（Stored Function）等，如图 3-3 所示。其中表是最基本的数据库对象，用于存放数据，视图是一种虚拟的表，存储过程和存储函数用于存放 SQL 代码，如同 C++、Java 或 Python 语言的函数。

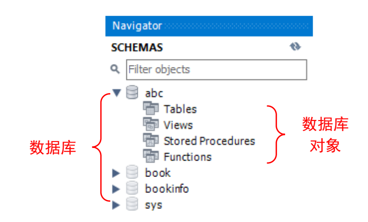


图 3-3 数据库对象

在使用数据库中的数据库对象之前，需要打开数据库，后续所有操作将在打开的数据库中进行。打开数据库的命令格式如下。

```
Use 数据库名;
```

例如下述命令打开 abc 数据库。

```
Use abc;
```

执行的结果如图 3-4 所示。

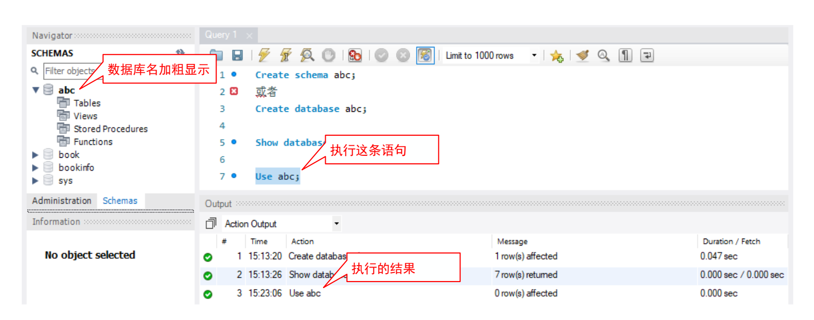


图 3-4 打开数据库

成功执行 Use abc;后，导航栏中数据库 abc 的名称以加粗的形式显示，表示这是当前打开的数据库，后续执行的 SQL 语句（创建表、插入数据、查询数据等语句）将在这个数据库完成。



MySQL Workbench 还提供一个便捷功能，双击导航栏上的数据库名称，使其以加粗的形式显示，表示切换该数据库为当前打开的数据库，与 Use 命令等效。

3.2.5 变更数据库

变更数据库的 SQL 语句的语法格式如下所示。

```
Alter {database | schema} 数据库名
      [default character set 字符集 | collate 校对];
```

参数说明参见 Create database 语句的参数说明。

变更数据库语句的主要用途是改变字符集或校对的设置，一般很少使用。



MySQL 早期版本的变更数据库语句曾提供过数据库改名的功能，出于安全方面的原因，该功能已被取消。

3.2.6 丢弃数据库

丢弃数据库的 SQL 语句的语法格式如下所示。

```
Drop {database | schema} [if exists] 数据库名;
```

参数说明如下。

- database | schema：必须二者取一，建议使用 database。
- if exists：可选的。省略时，如果要丢弃一个不存在的数据库，会提示出错。加上这个选项则可以避免这种出错提示。



将 drop 翻译为“丢弃”而不是“删除”，是想与 delete 的翻译“删除”有所区别。

例如下述语句丢弃一个名为 book 的数据库（在单元 1 创建的）。

```
Drop database book;
```

如果数据库 book 是存在的，该语句可以正常执行，如果数据库 book 不存在，该语言会提示出错。如果改为下述语句。

```
Drop database if exists book;
```

则不论数据库 book 存在与否，都不会出错。



丢弃数据库将丢弃（删除）数据库中的所有数据库对象，包括所有表以及表中的数据、视图、存储过程和存储函数，需谨慎操作。

3.3 表的创建与维护

表代表现实世界中的实体，在关系模型中称为关系。在创建表时，需要定义表中的所有列，然后在使用表时，向表插入一行一行的数据，数据库中的所有数据都是以这种形式存放在每一张表中的。

创建与维护表的 SQL 命令关键字也是 Create, Alter, Drop，它们可以分别用来创建、变更和丢弃表，在开始之前，先讲解 MySQL 的数据类型。

3.3.1 数据类型

在创建表时，需要指定列的数据类型。数据类型是指在数据库中保存的数据所采用的格式和占用的空间大小，列的数据类型对数据库有下述两方面的影响。

- 限制列能够保存的数据类型。例如，整数类型的列就不能保存实数；日期类型的列只能保存日期（年-月-日或年/月/日），而不能保存时间（时:分:秒）。
- 选择适当的列类型可以提高数据库的运行效率、降低资源消耗，例如减少存储空间或内存空间的使用。

MySQL 支持多种数据类型，常用的类型可以分为四大类：整型、字符串、浮点型和精确浮点型、以及日期和时间类型。附录 A 是对 MySQL 常用数据类型的总结。

1. 整型

整型分为 5 种，微整型（tinyint）、短整型（smallint）、中整型（mediumint）、整型（int）和大整型（bigint），它们分别占用 1 字节、2 字节、3 字节、4 字节和 8 字节。每一种整数都有有符号（signed）和无符号（unsigned）之分，如果没有指定，则是有符号的。

最常用的是微整型（tinyint）和整型（int）两种。

- 微整型（tinyint）：通常用于保存范围很小的数值，取值范围是-128～127，无符号微整型（tinyint unsigned）的取值范围是 0～255。
- 整型（int）：取值范围是-20 亿～20 亿，无符号整型（int unsigned）的取值范围是 0 亿～40 亿。

例如整型可以保存全中国的人口数，要保存全世界的人口数，则要用大整型（bigint）。

2. 浮点型

常见的浮点型有单精度、双精度和精确浮点型三种，通常只使用精确浮点型。

1) 单精度和双精度浮点型

单精度（float）和双精度（double）浮点型的取值范围基本上都可以满足日常需要，不同的是精度不同，前者是 6～7 位精度，后者可以达到约 15 位精度。

由于其不能准确定义精度的位数，通常不建议使用，而是使用精确浮点型。

2) 精确浮点型

MySQL 还支持一种特殊的浮点型——精确浮点型（decimal 或 numeric，两者同义），这种浮点型是精确的，最高可以达到 65 位的精度，可以根据需求，灵活地定义精度的位数，因此适合金融、商业、生产、统计、科研等所有行业的应用。

定义精确浮点型时必须同时指定精度和小数位数，例如 decimal (19, 4)表示精度为 19，小数位数为 4，存储时占用的字节数将取决于精度和小数位数。指定不同的精度和小数位数不仅仅是影响数值本身的精度，还会影响内部的存储格式以及占用空间的大小。

国际公认的会计原则（GAAP）规定，货币必须至少包含 4 位小数，因此要用精确浮点型，例如 decimal (19, 4)。本书实例中货币的数据类型采用 decimal (9, 2)，只是为了显示上的方便。

3. 字符串类型

与字符串有关的类型有如下几种。

- 定长字符串 Char(n)：占用固定长度的空间，不论字符串的实际长度是多少，都是占用指定长度（此处 n 表示固定长度）的空间，可能存在浪费空间的现象，因此只用于极短的字符串。
- 变长字符串 Varchar(n)：根据字符串的实际长度，占用实际使用的空间，但是最长不能超过指定的长度（此处 n 表示最大长度）。
- 短文本（Tinytext）：类似于变长字符串，但内部处理机制不同，可以用 Varchar 替代。
- 文本（Text）：类似于变长字符串，但内部处理机制不同，可以用 Varchar 替代。
- 长文本（Mediumtext）：保存长文本，由于太长而不能参与排序。
- 极长文本（Longtext）：保存极长文本，由于太长而不能参与排序。

在实际开发中，最常使用的是 Varchar 类型。在选择类型时要注意，不同类型的查询速度是不同的，Char 最快，Varchar 次之，各种 Text 最慢。

字符串常量要用单引号引起来，例如字符串“It is me.”写为 'It is me.'，如果字符串中含有单引号，则需要用两个单引号来表示，例如字符串“It's me.”写为 'It''s me.'，注意字符串内是两个单引号而不是一个双引号。

从 MySQL 5.0 开始，汉字与英文字母同样计数为 1，例如，Varchar(5)最多可以保存 5 个英文字母或 5 个汉字。

在 MySQL 中，字符串还可以使用转义字符，用于表示一些特殊含义的字符，如表 3-4 所示。

表 3-4 常用转义字符

转义字符	说明	转义字符	说明
\'	单引号	\"	双引号
\n	换行符	\r	回车符
\t	制表符	\\	反斜线（\）本身

例如用转义字符表示单引号，可将字符串“It's me.”写为 'It\'s me.'，它与 'It''s me.' 是等价的。

在 MySQL 中，还可以使用双引号将字符串括起来，这时字符串内的单引号不需特别处理，例如将字符串“It's me.”写为 "It's me."，这种写法是不 SQL 的标准写法，不建议使用。

另外还有二进制类型，用于保存图片等二进制数据，与字符串一样，也分为定长、变长、短、中、长和极长 6 种，不书不作讨论。

4. 日期和时间类型

与日期和时间有关的类型有以下 5 种。

- 日期（Date）：只能保存日期（年-月-日或年/月/日），不能保存时间（时:分:秒）。
- 时间（Time）：只能保存时间（时:分:秒），不能保存日期（年-月-日或年/月/日）。
- 年份（Year）：只能保存年份，取值范围 1901 年到 2155 年。
- 日期时间（DateTime）：同时保存日期和时间（年-月-日 时:分:秒或年/月/日 时:分:秒），取值范围比较大，从公元 1000 年到 9999 年。
- 时间戳（Timestamp）：同时保存日期和时间（年-月-日 时:分:秒或年/月/日 时:分:秒），取值范围比较小，从 1970 年到 2038 年。但它有一个特点，同时保存了时区的信息，因此可以在不同的时区正确地处理时间。

日期和时间类型的常量的表示方式如表 3-5 所示。

表 3-5 日期和时间类型的常量的表示方式

类型	格式	例子	说明
日期	年-月-日或年/月/日	'2020/12/09'或"2020-12-09"	可用单引号或双引号括起来
时间	时:分:秒	'13:30:00'	默认采用 24 小时制
年份	整数或字符串	2020 或'2020'	范围 1901 到 2155 的整数或字符串
日期时间 和时间戳	年-月-日 时:分:秒或 年/月/日 时:分:秒	'2020-12-09 13:30:00'或 "2020/12/09 13:30:00"	日期和时间之间用空格分隔 时间戳还要加上时区信息

3.3.2 创建表

1. 语法格式

创建数据库的 SQL 语句的语法格式如下所示。

```
Create table [if not exists] 表名 (            -- 表名后有一个圆括号，表示列定义的开始
    列名 1 数据类型 1 [列级约束 1],            -- 每个列定义结束时加逗号“,”作为分隔
```

```
    列名 2  数据类型 2 [列级约束 2],
    …
    列名 n  数据类型 n [列级约束 n]          -- 最后一个列定义结束时不能加逗号“,”
);                                           -- 最后是一个反圆括号，以分号结束整条 SQL 语句
```

参数说明如下。

- 表名：将要创建的表的名字，在同一个数据库中，表名必须唯一。
- 列名：表中列的名字，在同一张表中，列名必须唯一。
- 数据类型：设置该列数据的格式和存储方式，见前一小节和附录 A。
- 列级约束：可选的，指定该列的约束，见表 3-6。
- if not exists：与创建数据库的同名参数意义相同。

创建表还有更多的选项，例如指定存储引擎在单元 8 讲解，其他选项本书不予讲解。

2. 常用列级约束

表 3-6 常用列级约束

列级约束	约束类型	说明
primary key	主键约束	指定该列为主键，一张表只能有一个。整型的主键可加上自增量（auto_increment）。
references 主表(id)	外键约束	通常是在创建表之后再单独创建，将在“3.3.4 外键约束”讲解。
not null 或 null	非空约束	不允许为空或允许为空。主键列必须加上 not null（这是强制的）。
unique	唯一性约束	强制该列的数据不允许重复，例如身份证号。主键已经具有唯一性约束。
default	默认约束	插入数据时，如果该列的值为空，则以默认约束的参数作为该列的值。
check	检查约束	插入或更新数据时，检查数据是否符合要求。例子见“3.4.6 检查约束”

表的创建必须严格按照数据结构设计的成果来进行，包括表名、列名、数据类型、各种约束，甚至是注释等细节都必须严格一致，不允许有任何差错。

3. 创建一张简单的表

Jitor 实训 | 附录 C 实训 3-1

例如下述语句在数据库 abc 中创建一张名为 test 的表，该表演示了几种典型的数据类型，以及不允许为空、唯一性约束和默认值的使用。

```
Drop database if exists abc;  -- 删除数据库 abc
Create database abc;          -- 创建数据库 abc
Use abc;                      -- 打开数据库 abc，后续操作在全新的数据库中进行

Create table test (
    id int primary key not null auto_increment,   -- 自增主键
    col_char char(6) unique not null,             -- 非空列，唯一性约束
    col_varchar varchar(45) null,
    col_decimal decimal(9,4) null,
    col_datetime datetime not null,               -- 非空列
    col_tinyint tinyint not null default 2,       -- 非空列，但有默认值（2）
    col_text text null
);
```

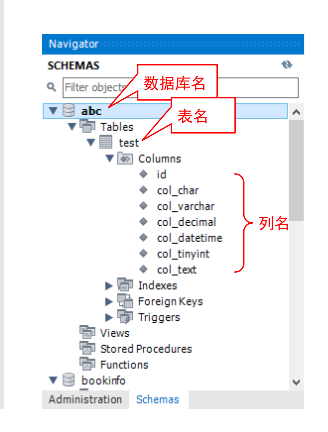


表 3-7 新创建的表

执行上述语句，将在数据库 abc 中创建表 test，需要刷新导航栏才能显示新创建的表 test，依次展开表名前的三角箭头，可以看到表中包含的列（Column）、索引（Index）、外键（Foreign Key）和触发器（Trigger）等数据库对象，如表 3-7 所示。

从上述创建表的语句中，可以反推出 test 表的数据结构，如表 3-8 所示。

表 3-8 test 表的数据结构

序号	列名	数据类型	为空性	唯一性	默认值	说明
1	id	int	not null	uqinue		主键，自增量的
2	col_char	char(6)	not null	uqinue		具有唯一性约束，可能用于编号，如学号、商品编号等
3	col_varchar	varchar(45)	null			Varchar 类型的列是最常用的
4	col_decimal	decimal(9,4)	null			4 位小数，可能用于货币
5	col_datetime	datetime	not null			必提供日期时间的值
6	col_tinyint	tinyint	not null		2	插入数据时，如果没有提供值，则置为默认值 2
7	col_text	text	null			Text 类型的列可能用于备注

这时通过图形界面向 abc 数据库的 test 表插入一些测试数据（输入数据的办法参见单元 1 的图 1-25），看看自增量的主键值是如何生成的，默认值是何时起作用的，输入了违反数据类型要求的数据时会有什么出错信息，违反了为空性、唯一性的数据又会有什么出错信息。如图 3-5 所示的是一条出错信息，意思是“第一行 col_char 列的数据太长”，因为该列的最大长度是 6，而输入的数据长度为 8。

```
1406: Data too long for column 'col_char' at row 1
```

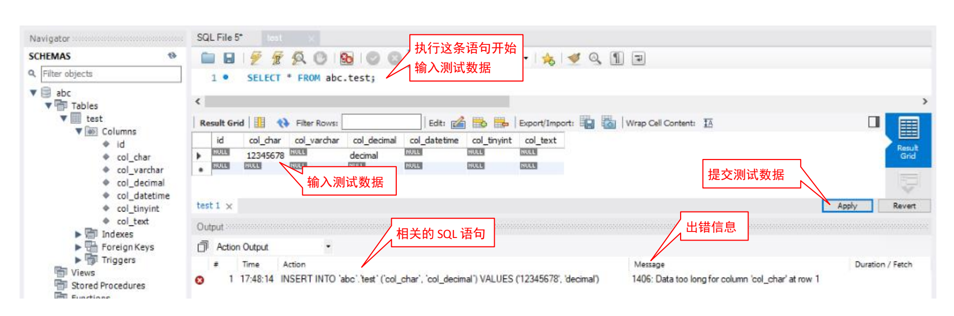


图 3-5 输入非法数据时的出错信息

4. 创建图书信息数据库的表



这一小节将创建单元 2 的“图书信息数据库”中的两张表，将数据结构（表 2-1 和表 2-2）改写为如表 3-9 和表 3-10 所示。

表 3-9 出版社表（publisher）的数据结构

序号	列名	数据类型	为空性	唯一性	默认值	备注
1	id	int	not null	uqinue		主键（ID），自增量
2	col_name	varchar(45)	null			出版社
3	col_addr	varchar(100)	null			地址
4	col_tel	varchar(45)	null			联系电话

表 3-10 图书表（book）的数据结构

序号	列名	数据类型	为空性	唯一性	默认值	备注
1	id	int	not null	uqinue		主键（ID），自增量
2	col_author	varchar(45)	null			作者
3	col_title	varchar(100)	null			书名
4	col_year	year	null			出版年份
5	col_isbn	varchar(45)	null			ISBN 书号

6	col_price	decimal(9,2)	null			价格
7	id_publisher	int	null			外键（出版社 ID）

然后编写创建出版社表和图书表的语句，如下所示。

```
Drop database if exists bookinfo;  -- 删除数据库 bookinfo
Create database bookinfo;          -- 创建数据库 bookinfo
Use bookinfo;                      -- 打开数据库 bookinfo，后续操作在全新的数据库中进行

-- 创建出版社表的语句
Create table publisher (
    id int primary key not null auto_increment,
    col_name varchar(45) default null,
    col_addr varchar(100) default null,
    col_tel varchar(45) default null
);

-- 创建图书表的语句
Create table book (
    id int primary key not null auto_increment,
    col_author varchar(45) default null,
    col_title varchar(100) default null,
    col_year year default null,
    col_isbn varchar(45) default null,
    col_price decimal(9,2) default null,
    id_publisher int default null
);
```

上述语句创建了出版社表（主表）和图书表（从表），从表（图书表）的外键参照主表（出版社表）的主键，虽然还没有创建外键约束，但参照联系是存在的。外键约束将在“3.3.4 外键约束”详细讲解。

关于列名的命名，可以采用如下的命名规范。

- 主键列的列名是 id，简洁明了，建议所有表的主键都命名为 id。
- 外键列的列名以 id_为前缀，后接被参照的主表表名。
- 其余列是普通列，列名前加上 col_作为前缀，以避免与关键字冲突。

3.3.3 变更表



附录 C 实训 3-3

在创建表之后，由于各种原因，可能需要变更（修改）表的结构，这时需要使用 Alter 语句来变更表的结构，语法格式如下所示。

```
Alter table 表名
    [add column 列名 列定义
    | add constraint 约束名 foreign key (外键名) references 主表名(主键名)
    | change column 旧列名 新列名 列定义
    | drop column 列名
    | drop constraint 约束名
    | rename to 新表名];
```

参数说明如下。

- add column：添加列，其中的列定义参见创建表（Create table）的参数。
- add constraint：添加约束，这里以外键约束为例。基于外键约束的重要性，将在下一小节详细讲解如何添加外键约束。
- change column：修改列。
- drop column：丢弃列。
- drop constraint：丢弃约束。

- rename to：重命名表名为新表名。

从 Alter table 的语法格式可以看到，变更表时可以对表进行添加列、修改列和丢弃列等操作，下面分别讲解。



变更表的结构与删除表后重新创建表有一个区别，变更表的结构可以保留表中原有的数据，例如修改列时数据不会改变。删除表会删除表中的数据，重新创建的表中没有数据。

1. 添加列

这是一个添加列的例子。

```
Drop database if exists abc;        -- 删除数据库 abc
Create database abc;                -- 创建数据库 abc
Use abc;                            -- 打开数据库 abc，后续操作在全新的数据库中进行

Create table test (                 -- 先创建一张表，然后变更这张表
    id int primary key not null auto_increment,
    col_char char(6) unique not null
);

Alter table test                    -- 在这张表上添加列
    add column col_varchar varchar(45) null;
Alter table test                    -- 继续添加列
    add column col_decimal decimal(9, 4) null;
```

上述例子中，最先创建的 test 表只有两列，主键列和 col_char 列，再先后向该表添加了两列 col_varchar 和 col_decimal 列。



如果添加的列有非空约束（not null），而表中已有数据，那么已有的行中新添加列的值为空，违反了非空约束的要求，就会因出错而失败，这时可以先添加允许空的列，然后修改该列所有行的数据为非空的某个值，再将该列改为不允许为空。

2. 修改列

这是一个修改列的例子，在前述的 test 表上继续变更该表。

```
Alter table test              -- 修改 col_varchar 列的最大长度为 500，新旧列名相同
    change column col_varchar col_varchar varchar(500) null;
Alter table test              -- 修改 col_decimal 列的列名为 col_price，小数位数改为 2，并且不允许为空
    change column col_decimal col_price decimal(9, 2) not null;
```



如果修改列时，表中已有数据会使之违反新的数据类型的要求，就会因出错而失败。例如缩短字符串的最大长度，而已有数据中存在超过新的最大长度的数据，或者是将字符类型改为整型或浮点型，而已有数据中存在无法转换为整型或浮点型的数据。

3. 丢弃列

这是一个丢弃列的例子，在前述的 test 表上继续变更该表。

```
Alter table test                    -- 丢弃 col_price 列。注意这个列是在上一步中从 col_decimal 改名而来的
    drop col_price;
```



丢弃列时，将丢弃该列的所有数据，需谨慎操作。

4. 表的重命名

例如下述语句将表 test 重命名为 test1，表的结构和数据保持不变。

```
Alter table test rename to test1;
```

3.3.4 外键约束

1. 创建外键约束的要求

创建外键约束有一个要求，外键与主键的数据类型严格一致。

2. 创建外键约束的办法

有三种创建外键约束的办法，如下所示。

- 创建表时，定义列级外键约束，如表 3-6 所示，但没有用例子加以讲解。
- 创建表时，定义表级外键约束，在“3.3.5 列出表及表结构”最后有一段代码含有这样的例子。
- 创建表后，为表添加外键约束，这种办法最为灵活，下面将进行讲解。

3. 为表添加外键约束

从附录 D 打开“项目 3a 公共代码共享”，从“单元 3 数据定义”将公共代码复制代码到 MySQL Workbench 运行，创建出版社表和图书表。



附录 C 实训 3-4

创建出版社表和图书表之后，再为图书表的外键（id_publisher）添加外键约束，代码如下。

```
Alter table book
    add constraint fk_book_publisher foreign key (id_publisher) references publisher(id);
```

其中外键名是 fk_book_publisher，fk 表示外键（foreign key），后接从表名和主表名。References 的词义是参照（也翻译为引用），要用动词的第三人称形式，加上后缀 s。

添加外键约束之后，如果向图书表插入一行，代码如下。

```
Insert into book () values (null, '作者 1', '书名 1', 2024, '223456789', 39.6, 1);
```

这时插入数据失败，显示如下出错信息，证明外键约束能够正常起作用。

```
Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`abc`.`book`, CONSTRAINT
`fk_book_publisher` FOREIGN KEY (`id_publisher`) REFERENCES `publisher` (`id`))
```

这是因为出版社表中还没有数据，图书表的外键参照了不存在的出版社表主键值 1。



如果主表还未创建，则外键约束也无法创建。因此通常的操作顺序是：第一步先创建所有表，第二步立即添加外键约束。正因为如此，为表添加外键约束是最为灵活的办法。

4. 丢弃外键约束

丢弃外键约束的代码如下，这时必须知道外键约束的名字。

```
Alter table book drop constraint fk_book_publisher;
```

如果不知道外键约束的名字，参考下一小节的讲解，可以查出外键约束的名字。



丢弃外键约束并不会丢弃外键，外键依然存在，并且外键的功能也存在，仅仅是输入了错误的外键值时，不会因为出错而失败。

3.3.5 列出表及表结构

1) 列出表的列表

列出当前数据库中所有表的语句如下所示。

```
Show tables;
```

例如下述代码将列出数据库 abc 中的所有表。运行结果如图 3-6 所示。

```
Use bookinfo;
Show tables;
```

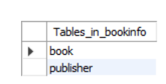
2) 列出指定表的表结构

列出指定表的表结构的语句如下所示。

```
Describe 表名;
```

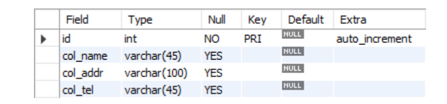
例如下述代码将列出出版社表的表结构。运行结果如图 3-7 所示。

```
Describe publisher;
```



Tables_in_bookinfo
book
publisher

图 3-6 列出所有表



Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
col_name	varchar(45)	YES		NULL	
col_addr	varchar(100)	YES		NULL	
col_tel	varchar(45)	YES		NULL	

图 3-7 列出出版社表的表结构

3) 列出创建表的 Create table 语句

如果要查看创建表所使用的 Create table 语句的代码，可以使用下述语句。

```
Show create table book;
```

运行结果如图 3-8 所示。

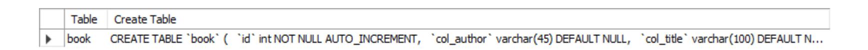


Table	Create Table
book	CREATE TABLE \`book\` (\`id\` int NOT NULL AUTO_INCREMENT, \`col_author\` varchar(45) DEFAULT NULL, \`col_title\` varchar(100) DEFAULT N...

图 3-8

其中的 Create table 语句内容如下，包括了 Alter tabler 所添加的外键约束。

```
CREATE TABLE `book` (
   `id` int NOT NULL AUTO_INCREMENT,
   `col_author` varchar(45) DEFAULT NULL,
   `col_title` varchar(100) DEFAULT NULL,
   `col_year` year DEFAULT NULL,
   `col_isbn` varchar(45) DEFAULT NULL,
   `col_price` decimal(9,2) DEFAULT NULL,
   `id_publisher` int DEFAULT NULL,
   PRIMARY KEY (`id`),
   KEY `fk_book_publisher` (`id_publisher`),
   CONSTRAINT `fk_book_publisher` FOREIGN KEY (`id_publisher`) REFERENCES `publisher` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

在这条 Create table 语句中，定义了表级外键约束，从而无需在创建表后，再添加外键约束。

3.3.6 复制表

复制表的 SQL 语句的语法格式如下所示。

```
Create table 新表名 like 原表名;
```

例如下述语句复制 test1 表（在“3.3.3 变更表”已经将 test 表改名为 test1 表）为 test2 表。两张表的结构是完全一模一样的，同时也复制了主键约束、外键约束和索引等，但不会将数据复制到新表中。

```
Use abc;
Create table test2 like test1;
```



如果想要在复制表后，再复制数据，请参见“6.1.3 子查询与增删改”。

3.3.7 丢弃表

丢弃表的 SQL 语句的语法格式如下所示。

```
Drop table [if exists] 表名;
```

例如下述语句丢弃名为 book 的表。

```
Use bookinfo;
Drop table if exists book;
```

由于含有 if exists，不论表 book 存在与否，都不会出错。



(1)丢弃表将丢弃（删除）表中的所有数据，需谨慎操作。

(2)如果丢弃一张主表，则会因为违反了外键约束而丢弃失败。应该先丢弃从表，再丢弃主表。

3.4 数据完整性约束

数据模型的三要素是数据结构、数据操作和数据完整性约束。关系数据库是严格按照关系模型要求建立的数据库，关系模型对数据结构、数据操作和数据完整性约束都有严格的要求。

- 关系模型的数据结构：这是关系模型的基础，关系模型由二维表组成，必须满足关系模型的 6 项基本特征，还应该满足 1NF、2NF 和 3NF 的要求，如果不满足，就需要拆分表，以达到三个原子性：属性、属性值和表的原子性，在单元 2 作过深入讲解。
- 关系模型的数据操作：这是关系模型的目的，使用数据库的目的就是为了对数据的增删改查，这部分将在单元 4 和单元 5 讲解。
- 关系模型的数据完整性约束：这是关系模型的灵魂，特别是主键和外键，体现了关系之间的联系，从单元 1 的例子起就已经陆续讲解过，本节对此作进一步的深入讲解。

3.4.1 实体完整性约束（主键约束）

实体完整性约束也称为主键约束，主键用于唯一标识表中的行，因此主键必须唯一，不能为空。

每张表有且只有一个主键，通常在创建表的同时创建主键，主键强制性地具有非空约束和唯一性约束。数据类型建议用整型，最好是自增量的，这样可以由数据库自动生成，用户无需输入，减少输入时的错误。

3.4.2 参照完整性约束（外键约束）

参照完整性约束也称为外键约束，它用于参照（引用）另一张表中行，因此它的值必须是被参照的主键的值的一个，即不能参照一个不存在的行。

一张表可以有零个、一个或多个外键，多个外键可以参照不同的表，也可以参照同一张表，例如一个订单的开单人、审核人、发货人都是公司的员工，这时都是参照同一张表，员工表。

外键可以有也可以没有唯一性约束，外键有唯一性约束时，表示联系是一对一的，没有唯一性约束时，表示联系是一对多的。

外键可以有也可以没有非空约束，根据用户需求来决定，外键不允许为空时，表示输入时就必须指定参照的行，外键允许为空时，表示输入时可以不指定参照的行，也许以后会补充输入，也许是不需要参照其他行。

3.4.3 唯一性约束

拥有唯一性约束的列就是候选键，根据用户需求而设置，例如学号列、身份证列和手机号码列都需要唯一性约束，以防止数据输入错误。

唯一性约束与主键约束的区别是，一张表中可以有多个唯一性约束，但只能有一个主键约束，唯一性约束可以为空或不为空，主键约束不允许为空。

唯一性约束允许为空时，只允许有一个空值，其他行如果又出现了空值，则违反了唯一性约束，因此唯一性约束通常是不允许为空的。

3.4.4 非空约束

非空约束就是允许为空或不允许为空，允许为空时，输入时可以不提供值，不允许为空时，输入为空

的值时会导致出错。

空是没有值，而不是 0 或空字符串，0 表示有值，值为 0，空串表示有值，值是长度为 0 的字符串。举个例子，某学生的考试成绩为空，表示该学生没有参加考试或成绩还未输入，而考试成绩为 0，表示该学生参考加考试，考试成绩为 0 分。

3.4.5 默认约束

默认约束就是插入行时，提供的值为空时，会用默认值来自动赋值。

一个列提供了默认值，输入时也提供了值，这时优先使用输入的值。

一个列同时提供了默认约束和非空约束时，输入时不提供值也不会导致出错，因为这时会用默认值作为输入的值。

3.4.6 检查约束

检查约束在数据类型、非空约束、唯一性约束的基础上，对属性的取值范围作进一步的约束。例如，如果学生成绩表的成绩列是微整型（TinyInt），则取值范围是-128~127，而用户会要求它的取值范围是 0~100，这时可以通过检查约束来实现。

例如下述代码对成绩列定义了一个检查约束。

```
Create table score(
	id int primary key not null auto_increment,
	name varchar(16) not null,
	score int not null check(score>=0 and score<=100)
);
```

如果输入的成绩小于 0 或大于 100，都会出错，插入或更新失败，如图 3-9 所示。

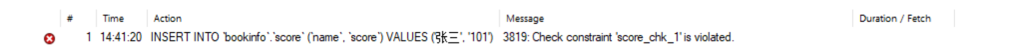


图 3-9 违反检查约束的出错信息

关系模型的六种数据完整性约束总结如表 3-11 所示。

表 3-11 六种数据完整性约束总结

数据完整性约束	作用	说明
实体完整性约束（主键约束）	用于唯一标识表中的行。它必须唯一、不能为空	建议用整型，自增量
参照完整性约束（外键约束）	用于参照另一张表的行。不允许参照不存在的行	先丢弃从表，后丢弃主表
唯一性约束	不允许出现重复值	变更时不得与已有数据冲突
非空约束	不允许出现空值	变更时不得与已有数据冲突
默认约束	插入行时，空值用它替代	变更时不得与已有数据冲突
检查约束	反映了用户的业务需求	变更时不得与已有数据冲突

数据完整性约束与数据定义和数据操纵有关，而与数据查询没有直接的关系，因为数据查询仅仅是使用数据，不会修改数据，因此不可能违反数据完整性约束。

但是数据查询需要有组织良好的数据，即一个“好”的关系数据库，而数据完整性约束就是“好”的关系数据库的基本保障，所以一定要重视数据完整性约束在数据定义和数据操纵时的作用。

3.5 建模工具的使用

使用 MySQL Workbench 创建数据库和表有如下三种途径。

● 使用图形界面：这是最简单的一种方式，适用于简单的项目，如单元 1 和单元 2 的例子所述。

● 使用 SQL 语句：这是通用性最好的一种方式，适用于任何场合，不论是使用图形界面还是使用建模工具，最终都是通过 SQL 语句实现的。

● 使用建模工具：这是一个专用的数据库模型设计工具，是设计人员的得力助手。

MySQL Workbench 提供的建模工具是专门用于设计 MySQL 的数据库模型，该工具将概念模型设计、关系模型设计和物理模型设计融为一体，以图形化的方式进行数据模型的设计，这个图形称为扩展 ER 图，这是建模工具的核心。

3.5.1 扩展 ER 图介绍

1. 扩展 ER 图

扩展 ER 图（EER，Exended ER 图）是 ER 图的一种，只是表现形式不同，EER 图还包含了更多的内容，不只是概念模型的信息，还包含关系模型和物理模型的信息，因此可以从 EER 图直接生成数据库的数据结构。

例如，如单元 2 图 2-8 所示的 ER 图（见“2.3.2 ER 模型”一节）可以画成 EER 图，如图 3-10 所示。

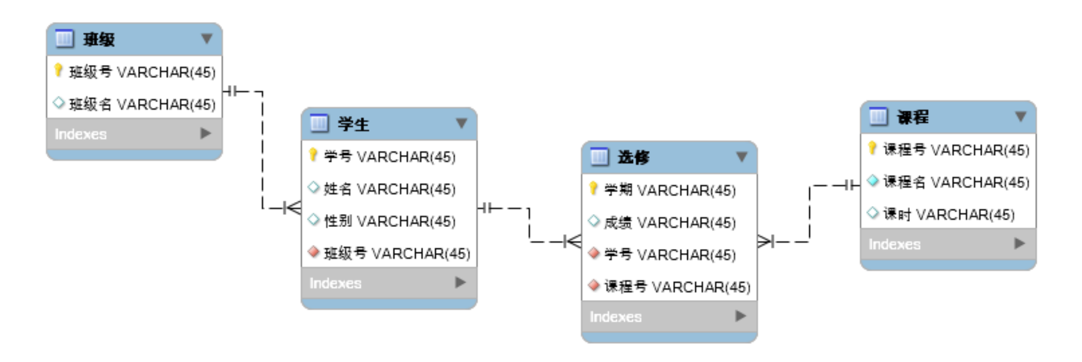


图 3-10 班级、学生和课程的 EER 图

从图 3-10 可以看到，它有概念模型、关系模型和物理模型的所有信息。

● 概念模型：拥有 ER 图的特点，实体（表）以矩形框表示，实体名（表名）写在标题上，属性（列）写在矩形框内，每个属性名（列名）一行，联系以折线表示，多的一方的未端用三分叉来表示。

● 关系模型：标识主键（列名前的钥匙图标）、外键（列名前菱形的颜色为红棕色）等关系模型的信息，将多对多联系转换为三张表，表示联系的表（图中是“选修”表）必须有两个外键，分别参照原来的两个实体（“学生”表和“课程”表）。

● 物理模型：添加了与具体的 DBMS（即 MySQL）相关的物理参数，如列类型、非空性、唯一性、索引（Indexes），以及物理存储要求等。图 3-10 中的列类型统一用 varchar(45)是因为图 2-8 的 ER 图缺少这类信息，这里用默认的数据类型代替。

单元 2 讲解过关系数据库设计的 6 步实施法，讲解过将概念模型、关系模型和物理模型三个设计阶段合并起来同时进行，MySQL Workbench 的建模工具也是这样做的，EER 图就是实现这种设计方法的一个工具。在绘制 EER 图时，要求深刻理解关系模型，完全理解 1NF、2NF、3NF，掌握关系数据库设计 6 步实施法，这样才能设计一个“好”的关系数据库。

2. 简化 EER 图

从 EER 图还可以方便地画出简化 EER 图，单击 EER 图中每张表右上角的三角形箭头，折叠表的详

细信息，就成为了简化 EER 图，如图 3-11 所示，这种简化 EER 图对于拥有数百个实体的大型项目是非常有用的。

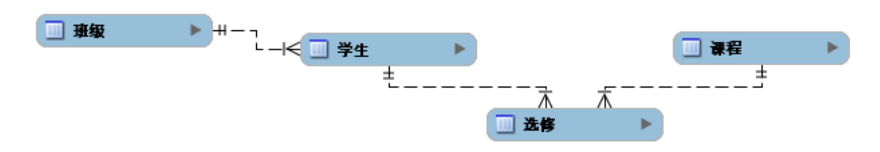


图 3-11 简化 EER 图

因此，ER 图主要用于理论阐述，实际工作中都是使用 EER 图来进行设计的，其他的数据库建模工具也有类似于 EER 图的表现形式。

3.5.2 使用建模工具

本节以单元 2“图书信息数据库”来讲解建模工具的使用，要求是创建出与“3.3 表的创建与维护”一节用 SQL 语句创建的数据库 bookinfo 完全相同的数据库。

- 数据库名：bookinfo1，这里使用 bookinfo1 是为了与“3.3 表的创建与维护”创建的 bookinfo 数据库相区别，前述的 bookinfo 数据库在本节最后的“逆向工程”还要用到的。
- 模型名称：bookinfo1，模型名称应该与数据库名相同。
- 表名：publisher 和 book，参见“3.3 表的创建与维护”的表 3-9 和表 3-10。

1. 打开建模工具

从 MySQL Workbench 首页添加一个模型，按照如图 3-12 所示的步骤操作，添加一个模型。



图 3-12 MySQL Workbench 首页

添加模型后，将自动打开建模工具，其默认界面如图 3-13 所示。

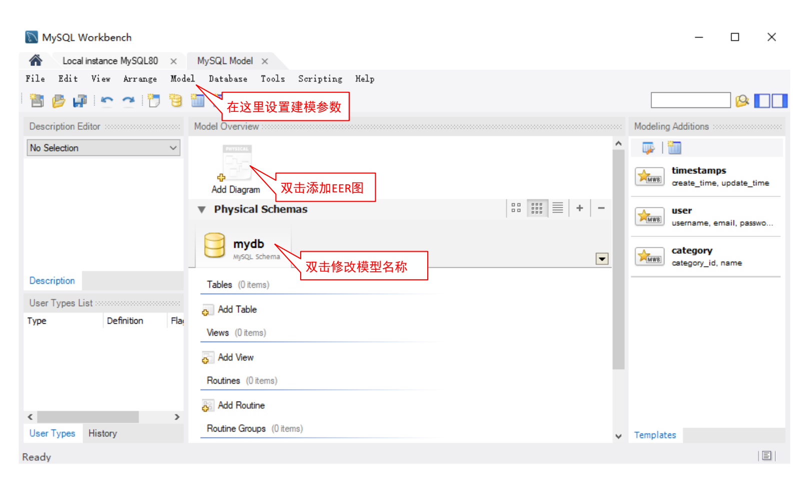


图 3-13 建模工具的默认界面

2. 设置建模参数

在开始设计之前，根据命名规范设置好建模参数，这里以下述命名规范为例设置建模参数。

- 主键列的名称统一为“id”。
- 外键列的名称为“id_被参照的表名”，例如 id_lib_book。
- 其他列的名称以 col_为前缀，后接该列的英文名称，不要用拼音或拼音首字母。

从主菜单 Model 中选择 Model Options，打开模型选项设置对话框，如图 3-14 所示，修改下述三个参数的设置。

- 主键列名（PK Column Name）：修改为 id。
- 普通列名（olumn Name）：修改为 col_，在使用时再修改为具体的名称。
- 外键列名（FK 的 Column Name）：修改为 id_%table%，将会自动用主表名替换%table%。

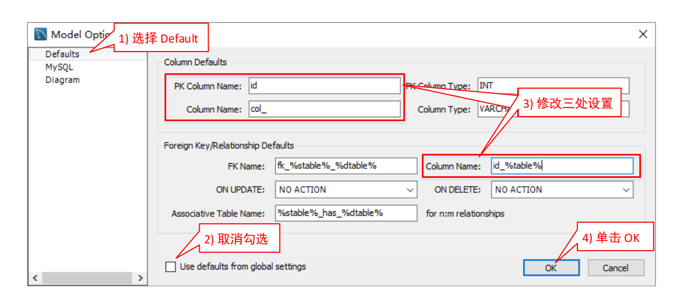


图 3-14 模型选项设置对话框

3. 修改模型名称

新添加的模型的默认名称是 mydb，双击该名称，将打开一个选项卡，按照如图 3-15 所示的步骤修改模型名称，可选的还能设置数据库的默认字符集。

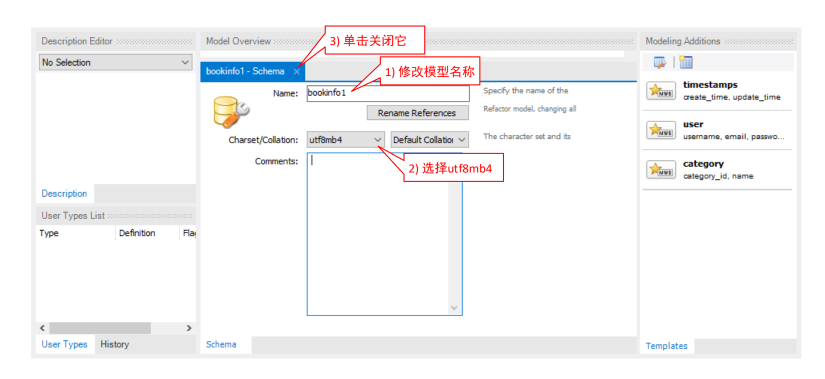


图 3-15 修改模型名称

4. 添加 EER 图

在图 3-13 所示的界面上，双击“Add Diagram”图标，添加 EER 图，这时打开 EER 图的设计界面，如图 3-16 所示。

图中需要关注的按钮有如下几个。

- “保存”按钮：将模型保存到文件中，文件后缀是.mwb。
- “添加表”按钮：单击该按钮后，在工作区的空白区域再单击一次，将添加一个表的图标。
- “添加一对一”按钮：先单击一对一的从表，后单击主表，将建立一对一联系。
- “添加一对多”按钮：先单击一对多的从表，后单击主表，将建立一对多联系。

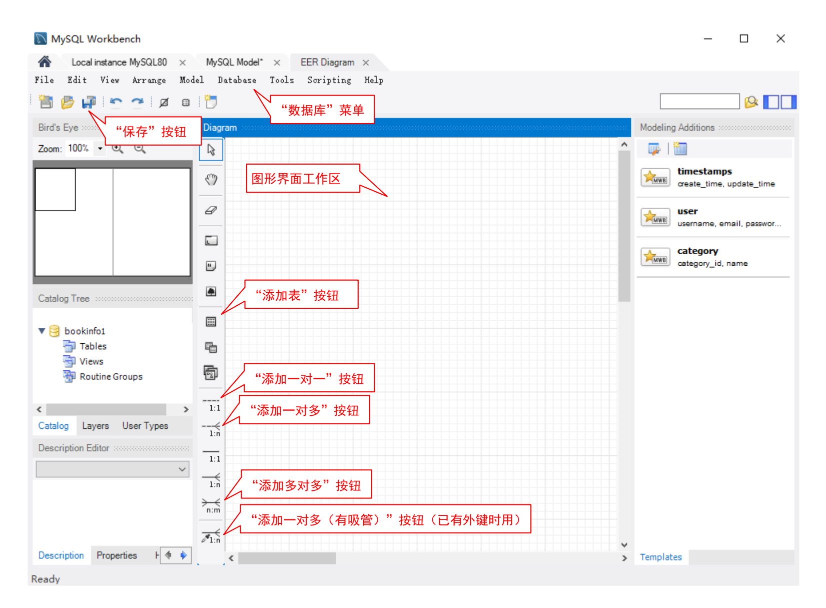


图 3-16 EER 图设计界面及有关按钮

- “添加多对多”按钮：先单击多对多的任何一方，后单击另一方，将在多对多双方之间自动新建一张表，这张表有两个外键，分别参照多对多的双方，将在“3.5.4 多对多联系”讲解。
- “添加一对多（有吸管）”按钮：前述创建联系的按钮会自动添加外键，而这个按钮是用已有的外键来创建联系，将在“3.5.5 建模工具使用技巧”讲解。

如图 3-16 所示的按钮还有另外一组“一对一”和“一对多”添加按钮，它们有一些区别，目前只需要使用最前面的两个。

5. 设计数据库

1) 理解用户需求

现在开始在 EER 图中设计数据库 bookinfo1 的两张表 publisher 表和 book 表，并建立它们之间的联系。设计数据库的目的是设计一个“好”的关系数据库，因此在开始之前，必须做到下面两点。

- 要充分理解用户的需求，详细分析用户的需求，才能设计出满足用户的数据库。
- 要深刻理解关系模型，要完全理解 1NF、2NF、3NF，掌握关系数据库设计 6 步实施法。

2) 添加表

单击“添加表”按钮，然后在工作区的空白区域再单击一次，添加一张表，重复一次，添加第二张表。这时的 EER 图工作区有两个表的图标，如所图 3-17 示。

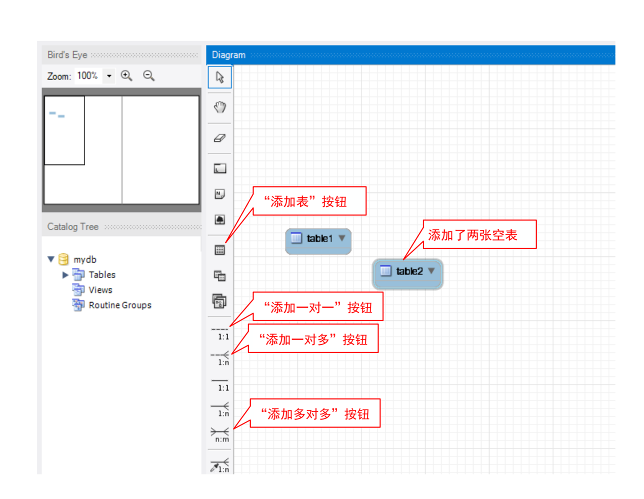


图 3-17 添加两张空表后的工作区

3) 修改表名和添加列定义

双击其中一个表图标，在工作区下方出现设计表的界面（类似于单元 1 的图 1-22），根据表 3-9 出版社表（publisher）的数据结构设计，修改表名，输入各列的定义，如图 3-18 所示。

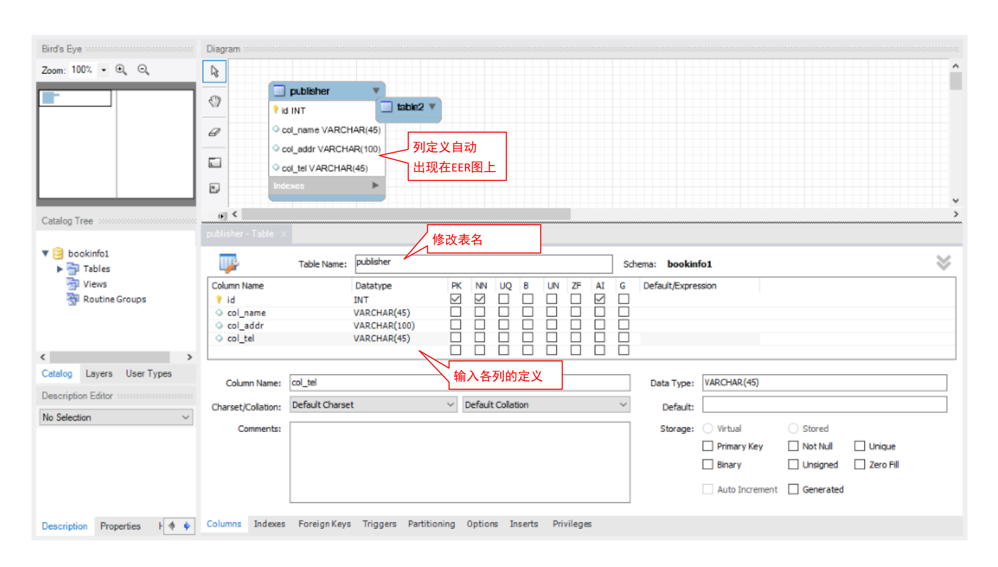


图 3-18 设计 publisher 表

添加第一列时，列名为 id，这是主键，其命名是在设置建模参数时指定的，主键默认选中 PK（主键）和 NN（非空）两个选项，而 AI（自增量）选项则需要根据设计来决定，在本书中，全部勾选 AI。其他的普通列命名为 col_，需要在这个前缀后加上具体的列名。

按同样的方式，根据表 3-10 图书表（book）的数据结构设计，设计 book 表，注意在这一步中，外键列不要输入，因为在下一步会自动添加外键列。建立联系前的出版社表和图书表如图 3-19 所示，注意图书表没有外键列。

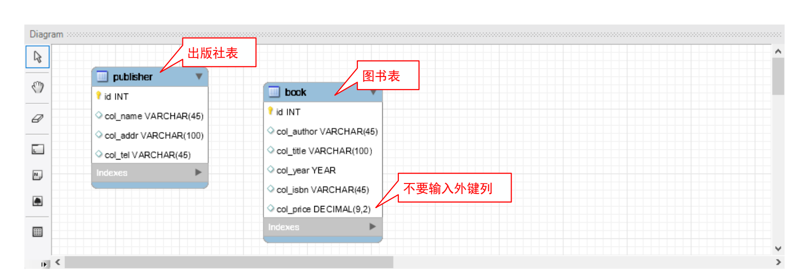


图 3-19 建立联系前的出版社表和图书表

4) 建立联系

使用“添加一对多”按钮来创建一对多联系，单击“1:n”按钮（有两个“1:n”按钮，选用前面的一个，虚线的），这时该按钮为选中状态（被蓝色方框包围），再单击图书表（从表）一次，接着单击出版社表（主表）一次，按钮恢复为未选中状态，这时 EER 图出现两个变化，一是增加了一条表示联系的折线，二是图书表自动添加了外键列，列名是根据设置建模参数时指定的外键命名规则命名的，如图 3-20 所示。



图 3-20 建立联系后的出版社表和图书表

如图 3-20 所示的是数据模型的 EER 图，可以导出为 png 格式的图片，方法是选择主菜单的 File →

Export → Export as PNG…，导出的图片可在需要时使用，例如将图片插入到毕业设计论文中。

5) 保存模型

至此，数据库模型设计完成，单击“保存”按钮，为模型文件起一个名字，保存起来。这个文件是一个重要的开发文档，应该妥善保存。

6. 正向工程生成数据库

在主菜单中选择 Database 的 Forward Engineer，参见图 3-16。这时弹出 Forward Engineer to Database 向导，如图 3-21 所示。这个向导一共有 5 步，全部选择默认值，单击 Next 按钮直到完成即可。

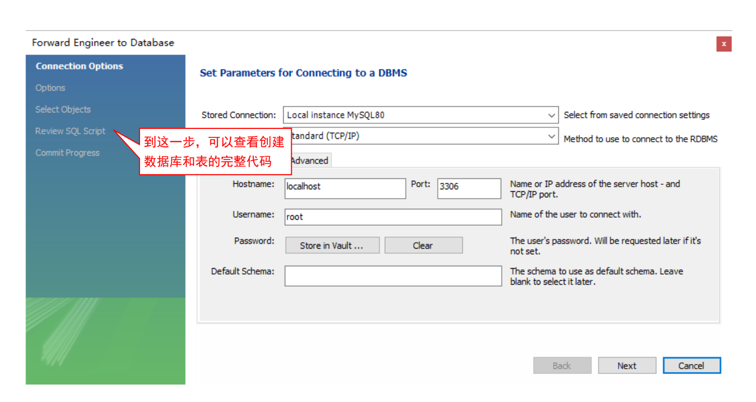


图 3-21 Forward Engineer to Database 向导的第一步

在第 4 步“Review SQL Script”可以看到创建数据库和表的完整代码，在这个例子中，生成的代码如下。

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR
_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';


-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
-- -----------------------------------------------------
-- Schema bookinfo1
-- -----------------------------------------------------


-- -----------------------------------------------------
-- Schema bookinfo1
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `bookinfo1` DEFAULT CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci ;
USE `bookinfo1` ;


-- -----------------------------------------------------
-- Table `bookinfo1`.`publisher`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bookinfo1`.`publisher` (
  `id` INT NOT NULL,
  `col_name` VARCHAR(45) NULL DEFAULT NULL,
  `col_addr` VARCHAR(100) NULL DEFAULT NULL,
  `col_tel` VARCHAR(45) NULL DEFAULT NULL,
```

```
  PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `bookinfo1`.`book`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bookinfo1`.`book` (
  `id` INT NOT NULL,
  `col_author` VARCHAR(45) NULL DEFAULT NULL,
  `col_title` VARCHAR(100) NULL DEFAULT NULL,
  `col_year` YEAR NULL DEFAULT NULL,
  `col_isbn` VARCHAR(45) NULL DEFAULT NULL,
  `col_price` DECIMAL(9,2) NULL DEFAULT NULL,
  `id_publisher` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_book_publisher_idx` (`id_publisher` ASC) VISIBLE,
  CONSTRAINT `fk_book_publisher`
    FOREIGN KEY (`id_publisher`)
    REFERENCES `bookinfo1`.`publisher` (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

这是一段非常规范的代码，由 MySQL Workbench 从数据模型自动生成。注意在代码中，外键约束是通过表级约束来创建的。

最后一步就是执行这段代码，创建数据库和表。完成后，返回到 Workbench 界面，刷新导航栏，可以看到正向工程生成的数据库 bookinfo1，以及数据库中的两张表，如图 3-22 所示。

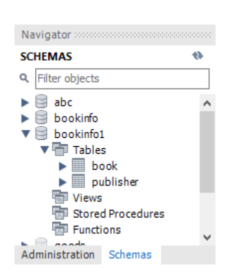


图 3-22 正向工程生成的数据库 bookinfo1

3.5.3 一对一和一对多联系

本小节以一个小例子，讲解一对一和一对多联系的创建，以及它们的区别。

如图 3-23 所示，有班主任表、班级表和学生表 3 张表，为方便理解，使用中文表名和中文列名，其中班主任表与班级表是一对一联系，班级表和学生表是一对多联系。

1. 一对一联系的创建

使用“添加一对一”按钮来创建一对一联系，单击“1:1”（有两个“1:1”按钮，选用前面的一个，虚线的），再分别单击一对一的双方，一对一联系创建完成。这时有一个细节需要考虑，即一对一的双方中

哪一方为主，在这个例子中，是班主任表为主，还是班级表为主？操作的次序是先单击为辅的一方，后单击为主的一方，打个不恰当的比喻，就像是儿子找父亲。如果先单击班级表，后单击班主任表，这时班主任表是主表，班级表是从表，从表将自动添加一个外键，参照主表（班主任表）的主键，这个外键还应该设置为唯一性约束（unique），如图 3-24 所示的班级表与班主任表。

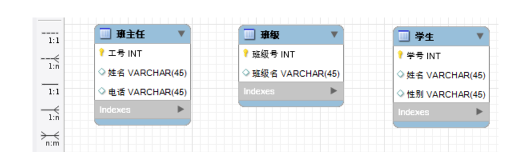


图 3-23 班主任、班级和学生 3 张表

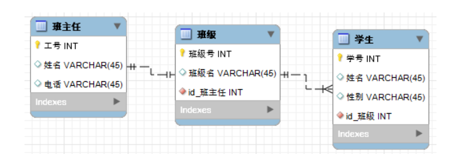


图 3-24 创建一对一、一对多联系

2. 一对多联系的创建

使用“添加一对多”按钮来创建一对多联系，单击“1:n”（有两个“1:n”按钮，选用前面的一个，虚线的），再单击多的一方（学生表，它是为辅的一方），最后单击一的一方（班级表，它是为主的一方）。打个不恰当的比喻，也是儿子找父亲，儿子可以有多个，父亲只有一个。多的一方（学生表）自动添加一个外键，参照一的一方（班级表）的主键，如图 3-24 所示的学生表与班级表。

比较图 3-24 所示的一对一和一对多的连接线，最主要的区别是，一对多联系中多的一方是三分叉的，这种线也称为鹬脚线，因此这种图也称为鹬脚图。

3.5.4 多对多联系

在前述“图书信息数据库”以及上一小节中都没有多对多联系的例子，在建模工具中创建多对多联系有两种办法。本小节以“3.5.1 扩展 ER 图介绍”图 3-10 所示的 EER 图来分别讲解这两种办法。

如图 3-25 所示，有班级表、学生表和课程表 3 张表，其中班级表和学生表是一对多联系，图中这个联系已经建好，学生表和课程表是多对多联系，下面分别用两种办法来创建。

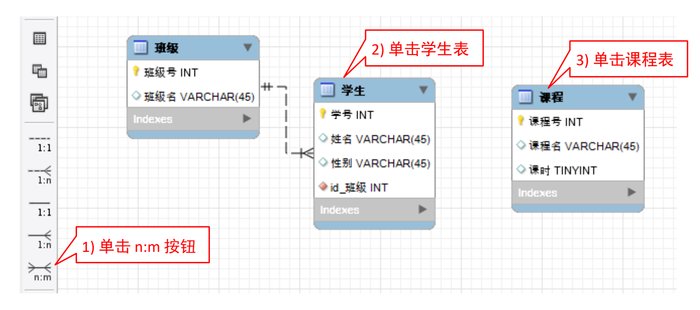


图 3-25 班级、学生和课程 3 张表

1. 直接创建多对多联系

这种办法是直接使用“添加多对多”按钮来创建多对多联系。在图 3-25 中，先单击“n:m”添加多对多按钮，再先后单击学生表和课程表，这时会自动创建一张新表，根据单击学生表和课程表的先后次序的不同，新表的名字是“学生_has_课程”或“课程_has_学生”，新表有两个外键，分别参照学生表和课程表，如图 3-26 所示。

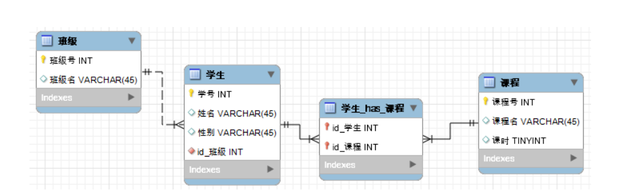


图 3-26 直接创建多对多联系

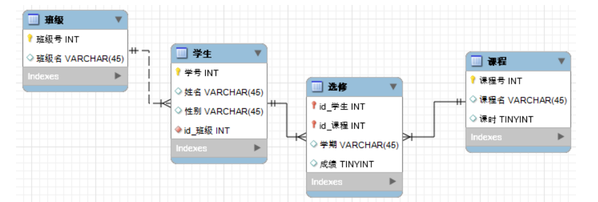


图 3-27 修改表名和添加属性

这时再修改表名，添加联系的属性，如图 3-27 所示，多对多联系创建完成。

2. 间接创建多对多联系

这种办法是先添加一张选修表，如图 3-28 所示，然后再为选修表创建两个多对一联系，如图 3-29 所示，多对多联系创建完成。

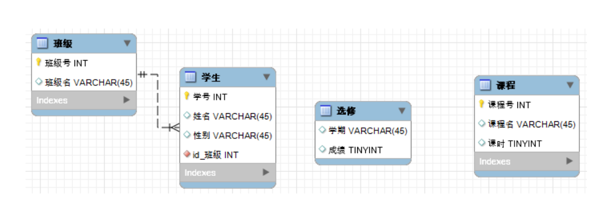


图 3-28 先添加选修表

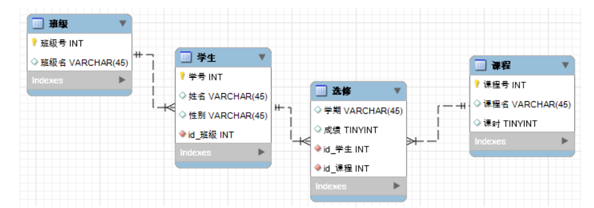


图 3-29 为选修表创建两个多对一联系

上述两种办法的结果完全相同，可以自由选用。

3.5.5 建模工具使用技巧

使用建模工具进行数据库设计时，可以参考单元 2“2.5 关系数据库设计 6 步实施法”，从用户需求中分辨出实体（表），建立表与表之间的联系，需要满足 1NF、2NF、3NF 的要求，即属性值的原子性和表的原子性的要求，必要时就要拆分表。



建模工具的核心功能就是建表、建立表之间的联系，需要根据规范化设计的要求来设计。因此单元 2“2.4 关系数据库设计”和“2.5 关系数据库设计 6 步实施法”是最重要的指导思想。

1. 拆分表

拆分表之前可能已经对表添加了许多列，这些列将分别属于拆分后的两张表中，为避免再次添加这些列，可以复制表，然后在复制的表中分别删除不属于自己的列，从而成为两张表，再在这两张之间建立一对一或一对多联系。

在表的右键菜单上选择 copy，然后在空白处的右键菜单上选择 paste，复制（Duplicate）一张表，新表的表名加上了_copy1，移动一下新表的位置，就能看到旧表和新表，如图 3-30。

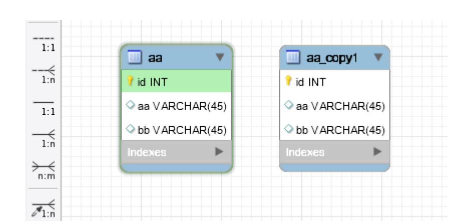


图 3-30 复制被拆分的表

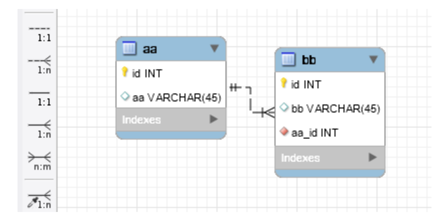


图 3-31 拆分后的两张表

然后删除两张表中不属于自己的属性，并立即建立两张表的联系，如图 3-31 所示，表拆分完成。

2. 已有外键时创建联系

前述创建一对一和一对多联系时将自动添加外键，如果外键列已经手工创建，这时可以用另一种按钮（位于工具栏的最后一个，需要关闭下方的表设计窗口才能看见），如图 3-32 所示。操作顺序是先单击从表的外键，然后单击主表的主键，注意这时要单击的是外键或主键的列名，而不是整张表。

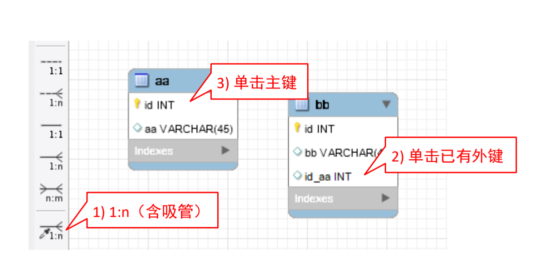


图 3-32 创建联系（已有外键列时）

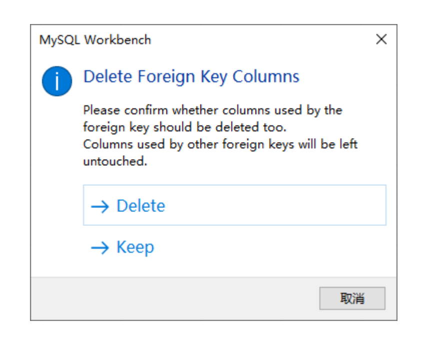


图 3-33 删除外键约束的两个选项

3. 删除外键约束

有时想要删除外键约束，通过外键约束连线的右键菜单删除外键约束时，会弹出一个对话框，提供删除的两个选项，如图 3-33 所示。这两个选项如下，根据需求选用。

- Delete：删除外键约束的同时，也删除从表的外键。
- Keep：只删除外键约束本身，保留从表的外键。

4. 如何隐藏设计细节

如果工作区上表的数量很多，或者是表的属性很多，这时可以单击表右上角的小三角图标，折叠或展开表的属性。折叠后隐藏细节，便于对全局有一个全面的认识。全部表都折叠后，就成为简化 EER 图，如图 3-11 所示。

3.5.6 正向工程和逆向工程

前述“图书信息数据库”例子是从数据模型生成数据库，这个过程称为正向工程（Forward Engineer），MySQL Workbench 还支持逆向工程（Reverse Engineer），即从数据库逆向生成数据模型。

这里以“3.3 表的创建与维护”一节用 SQL 语句创建的数据库 bookinfo 为例来讲解。

在主菜单中选择 Database 的 Reverse Engineer。这时弹出 Reverse Engineer Database 向导，如图 3-34 所示。这个向导一共有 7 步，仅在第 3 步 Select Schemas，勾选要作逆向工程的数据库，其他的步骤全部选择默认值，单击 Next 按钮直到完成即可。

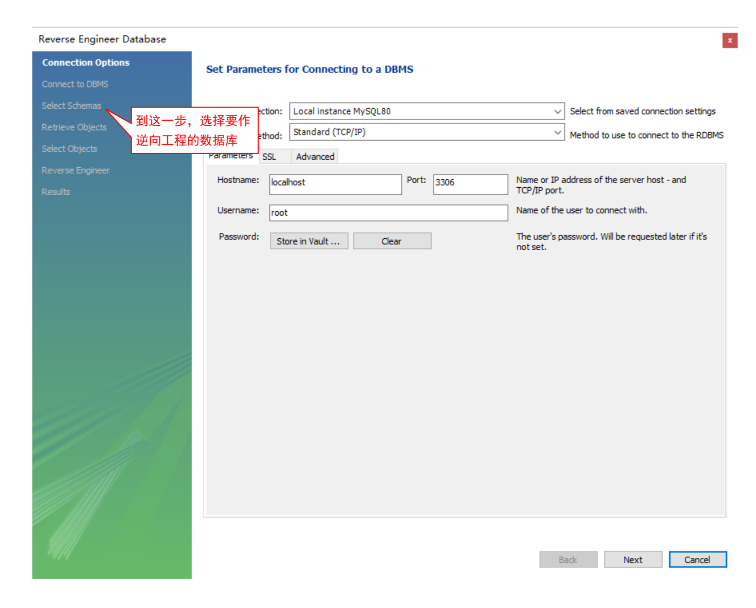


图 3-34 Reverse Engineer Database 向导的第一步

完成后，将自动打开由逆向工程生成的数据模型（EER 图），这个模型可以用作以下用途。

- 根据用户新的需求，对模型进行修改，然后通过正向工程，重新生成数据库，这对于项目的迭代更新非常有用，特别是在缺少原设计资料的情况下更加显示出逆向工程的巨大作用。
- 研究已有项目的数据库设计经验，可以从生成的模型中看到设计的细节，对于提高自己的设计水平有极大的帮助，例如单元 9 案例讲解“进销存管理系统”有 13 张表，通过逆向工程可以生成数据模型，学习该项目的设计经验。

【案例讲解】创建书店管理数据库



参考附录 D 的实战项目 2c，根据单元 2【案例讲解】“书店管理项目”的设计成果，为该项目创建书店管理数据库，要求如下。

- 数据库名：bookstore（模块名为 bs）
- 表结构：见单元 2 的表 2-22～表 2-26。

分别采用 SQL 语句和建模工具创建书店管理数据库。

任务 1 使用 SQL 语句创建书店管理数据库



在 Jitor 校验器中打开实训 3-5，该实训提供了所有列名等信息供复制粘贴使用，可以避免拼写错误。

1) 创建数据库

创建数据库 bookstore 的 SQL 语句如下。

```
Drop database if exists bookstore;        -- 删除数据库 bookstore
Create database bookstore;                -- 创建数据库 bookstore
```

2) 创建表

根据单元 2【案例讲解】“书店管理项目”的设计成果，创建 5 张表的 SQL 语句如下。

```
Use bookstore;                            -- 打开数据库 bookstore，后续操作在这个数据库中进行

Create table bs_publisher (
	id int primary key not null auto_increment,
	col_name varchar(50) not null default '',
	col_addre varchar(50) not null default '',
```

```
	col_tel varchar(50) not null default ''
);

Create table bs_book (
	id int primary key not null auto_increment,
	col_author varchar(50) not null default '',
	col_title varchar(50) not null default '',
	col_year year not null default '2155',
	col_isbn varchar(50) not null default '',
	col_classification varchar(50) not null default '',
	col_price decimal(13,2) not null default '0.00',
	id_bs_publisher int not null default '0'
);

Create table bs_customer (
	id int primary key not null auto_increment,
	col_name varchar(50) not null default '',
	col_sex char(1) not null default '',
	col_tel varchar(50) not null default '',
	col_addre varchar(50) not null default ''
);

Create table bs_order (
	id int primary key not null auto_increment,
	id_bs_customer int not null default '0',
	col_addre varchar(50) not null default '',
	col_total_amount decimal(13,2) not null default '0.00',
	col_status varchar(50) not null default '',
	col_order_date datetime not null default '2155-12-31 23:59:58',
	col_shipping_date datetime not null default '1901-01-01 00:00:00',
	col_remark text
);

Create table bs_order_detail (
	id int primary key not null auto_increment,
	id_bs_order int not null default '0',
	id_bs_book int not null default '0',
	col_price decimal(13,2) not null default '0.00',
	col_quantity int not null default '0',
	col_amount decimal(13,2) not null default '0.00'
);
```

3) 添加外键约束

添加外键约束的 SQL 语句如下。

```
Alter table bs_book
	add constraint fk_bs_book_bs_publisher foreign key (id_bs_publisher) references bs_publisher(id);

Alter table bs_order
	add constraint fk_bs_order_bs_customer1 foreign key (id_bs_customer) references bs_customer(id);

Alter table bs_order_detail
	add constraint fk_bs_order_detail_bs_order1 foreign key (id_bs_order) references bs_order(id);

Alter table bs_order_detail
	add constraint fk_bs_order_detail_bs_book1 foreign key (id_bs_book) references bs_book(id);
```

任务 2 使用建模工具创建书店管理数据库

使用建模工具进行书店管理数据库的数据建模，得到的 EER 图如图 3-35 所示。

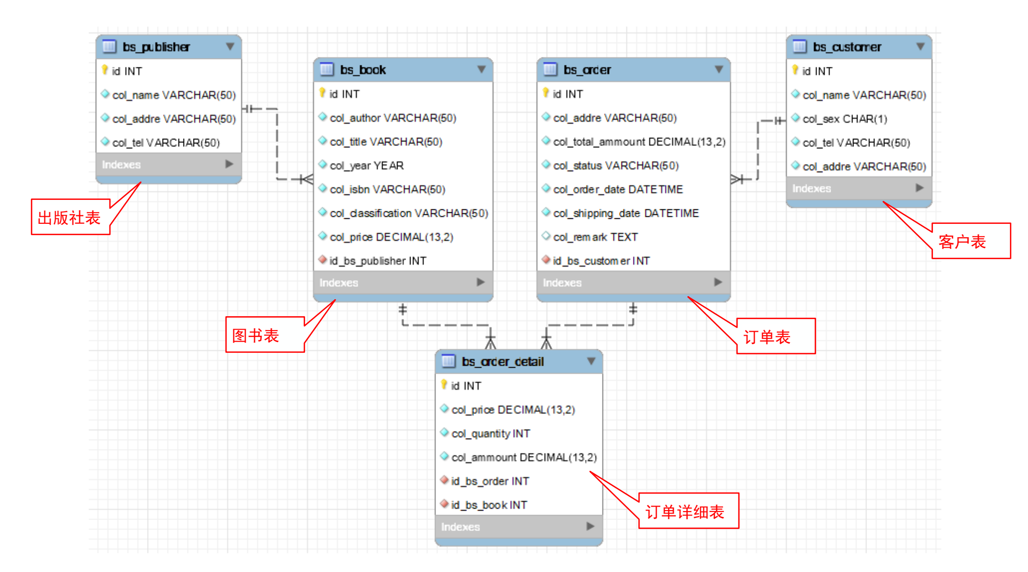


图 3-35 书店管理数据库的 EER 图

通过正向工程，可以从这个数据模型生成书店管理数据库，其结果与使用 SQL 语句创建数据库是完全相同的。



也可以通过逆向工程从前述 SQL 语句创建的书店管理数据库中生成图 3-35 所示的 EER 图。

【实战演练】创建图书借阅数据库



参考附录 D 的实战项目 2d，根据单元 2【实战演练】“图书借阅项目”的设计成果，为该项目创建图书借阅数据库，要求如下。

- 数据库名：library。
- 表结构：见单元 2 读者自行设计的成果。

要求得到两项成果：（1）创建数据库和表的 SQL 语句；（2）数据模型的 EER 图。可以采用下述两种方式中的一种来同时得到这两项成果。

- 先编写创建数据库和表的 SQL 语句，生成数据库后，通过逆向工程得到数据模型的 EER 图；
- 先在建模工具中设计数据模型的 EER 图，在生成数据库的过程中复制自动生成的 SQL 代码，修改后作为创建数据库和表的 SQL 语句。

【单元重点】

单元小结参见本单元的【思维导图】，单元重点如下。

- 数据库的创建与维护：重点是创建和使用数据库。
- 数据类型：重点是 10 种常用的数据类型。
- 表的创建与维护：重点是创建表，变更表和添加外键约束。
- 数据完整性约束：主键约束、外键约束、唯一性约束、非空约束、默认约束、检查约束。
- 主键约束（即实体完整性约束）：需要加深理解。
- 外键约束（即参照完整性约束）：需要加深理解，还要理解外键和外键约束的区别。
- 建模工具：EER 图的概念及其画法，通过正向工程从数据模型生成数据库。

【课后思考】

一、选择题

1. 创建表的 SQL 语句是（　　）。

   A. Create database　　B. Create table　　C. Alter database　　D. Alter table

2. Unique 是一种（　　）。

   A. 主键约束　　B. 外键约束　　C. 唯一性约束　　D. 非空约束

3. Not null 是一种（　　）。

   A. 主键约束　　B. 外键约束　　C. 唯一性约束　　D. 非空约束

4. 保存身份证号的列的数据类型最合适的是（　　）。

   A. Bigint　　B. Char　　C. Varchar　　D. Text

5. 保存金额的列的数据类型最合适的是（　　）。

   A. Int　　B. Float　　C. Double　　D. Decimal

6. 主键列定义中不会用到哪个关键字（　　）。

   A. Primary Key　　B. Not null　　C. Auto_increment　　D. Default

7. EER 图包含了哪些方面的信息【　　　】。

   A. 数据模型　　B. 概念模型　　C. 逻辑模型　　D. 物理模型

8. MySQL Workbench 的 EER 图，用折线表示联系，表示多的一方的末端用（　　）表示。

   A. 十字形　　B. 二十字形　　C. 圆圈形　　D. 三叉形

9. 创建了外键约束之后，下述哪种情况会出错（　　）。

   A. 先丢弃主表　　B. 先丢弃从表　　C. 先删除从表所有行　　D. 上述都不会出错

10. 没有创建外键约束时，下述哪种情况会出错（　　）。

    A. 先丢弃主表　　B. 先丢弃从表　　C. 先删除从表所有行　　D. 上述都不会出错

二、填空题

1. 最重要的两种键是_________和_________。
2. 从数据模型生成数据库的过程称为_________，从数据库生成数据模型的过程称为_________。

三、思考题

1. 创建数据库和表与单元 2 的数据库设计有什么关系？
2. 外键和外键约束有什么区别？
3. 建模工具在项目开发中有什么作用？

【课外拓展】

1、 从网上查找并下载 MySQL8.0 参考手册（或从附录 E 的在线资源中下载），找到 Alter table 一节（小节标题是 13.1.9 ALTER TABLE Statement），了解变更表的更多功能。

2、 查找有关数据建模的文章，了解建模工具的功能与作用。

单元4 数据操纵

【学习目标】

知识目标	◆ 熟练掌握删除语句的使用。
◆ 理解插入、更新、删除操作与数据约束的关系。	◆ 掌握清空语句的使用。
能力目标	素质目标
◆ 熟练掌握插入语句的使用。	◆ 认识到规则的重要性，人人遵纪守法
◆ 熟练掌握更新语句的使用。	◆ 防止数据泄露，坚守专业道德

【思维导图】



【情景导入】

小明学完了单元 3，觉得收获很大，学习了编写 SQL 语句，创建数据库，创建表以及维护表，用 SQL 语句实现了单元 2 设计的数据结构，很有一番成就感。还学习了建模工具的使用，用建模工具来实现一对一、一对多和多对多的设计，非常方便而实用。数据库设计已经完成，现在小明想学习下一阶段的内容，就是操纵数据库中的数据，让我们和小明一起学习吧。

【知识储备】

单元 2 对书店管理项目进行了数据库的设计，单元 3 创建了该项目的数据库和表，本单元将把数据输入到数据库中，这些操作包括插入、更新和删除。

4.1 数据插入 Insert

向表添加新的数据就是向表插入新的行，使用 Insert 语句实现。

4.1.1 Insert 语句

Insert 语句的语法格式如下所示。

```
Insert into 表名 [(列名列表)]
        values (值列表);
```

参数说明如下。

- 表名：被插入数据的表名。
- 列名列表：被插入数据的列名列表，省略时，表示表中的所有列。
- 值列表：插入数据的值列表，用逗号分隔，与列名列表一一对应，个数、顺序、数据类型相同，并且其含义也应该相同，否则会将值插入到错误的列中。

数据需要按一定的格式表示，见表 4-1。注意字符型的值的长度不能超过列定义的最大长度。

表 4-1 值列表中的数据格式

类型	说明	例子
数字	直接表示，整数或带小数点的数据	5、1.23
字符串	单引号括起来，内含的单引号可用两个单引号替代。不建议使用双引号	'It is me.'、'It''s me.'、'It\'s me.'
日期时间	用单引号括起来，需要符合日期、时间或日期时间的格式	'2020-04-13 08:52:00'

详细的讲解参见单元 3 的“3.3.1 数据类型”一节。

4.1.2 Insert 语句的使用

Jitor 实训 | 附录 C 实训 4-1

这里以“3.3.2 创建表”的 abc 数据库的 test 表为例，向该表插入数据。

首先创建数据库和表，然后在这张表上进行插入操作。创建数据库和表的语句如下所示，下述代码也可从附录 D“项目 3a 公共代码共享”找到，从“单元 4 数据操纵”将其中本节的公共代码复制到 MySQL Workbench 运行即可。

```
Drop database if exists abc;  -- 删除数据库 abc
Create database abc;          -- 创建数据库 abc
Use abc;                      -- 打开数据库 abc，后续操作在全新的数据库中进行

Create table test (
        id int primary key not null auto_increment,     -- 自增主键
        col_char char(6) unique not null,               -- 非空列，唯一性约束
        col_varchar varchar(45) null,
        col_decimal decimal(9,4) null,
        col_datetime datetime not null,                 -- 非空列
        col_tinyint tinyint not null default 2,         -- 非空列，但有默认值（2）
        col_text text null
);
```

1. 省略列名列表，指定主键值

省略列名列表时（列的次序以定义时的次序决定），必须提供表中所有列的数据，包括主键值。下述语句向 test 表插入 1 行数据，执行结果如图 4-1 第 1 行所示。

```
Use abc;
Insert into test
        values (1, '字符串 1', '变长字符串 1', 1.21, '2024-03-21 09:08:27', 121, '说明内容 1');
```

2. 省略列名列表，主键值为 null，自动增量

当主键是自动增量时，主键的值可以指定为 null，这时将赋给自动生成的主键值，保证不会重复。下述语句向 test 表插入第 2 行数据，执行结果如图 4-1 第 2 行所示，自动生成的主键值是 2。

```
Insert into test
```

```
    values (null, '字符串 2', '变长字符串 2', '1.22', '2024-03-22 09:08:27', '122', '说明内容 2');
```

注意如果主键不是自动增量的，则不允许用 null 值来替代，而必须提供实际的值，这时会很不方便。



自动增量的主键值不一定是连续增量的，每一次失败的插入操作都会浪费一个生成的主键值，而使下一次插入操作使用一个新的主键值。

3. 列出所有列，主键值为 null，自动增量

列出所有列时，与省略列名列表类似，但要注意值的次序与列名的次序一致。下述语句向 test 表插入第 3 行数据，执行结果如图 4-1 第 3 行所示。

```
Insert into test (id, col_char, col_varchar, col_decimal, col_datetime, col_tinyint, col_text)
    values (null, '字符串 3', '变长字符串 3', '1.23', '2024-03-23 09:08:27', '123', '说明内容 3');
```

4. 列出所有列，但不含自动增量的主键

列出所有列，但不含自动增量的主键时，这时要按次序提供列名列表中各列的数据。下述语句向 test 表插入第 4 行数据，执行结果如图 4-1 第 4 行所示。

```
Insert into test (col_char, col_varchar, col_decimal, col_datetime, col_tinyint, col_text)
    values ('字符串 4', '变长字符串 4', 1.24, '2024-03-24 09:08:27', 124, '说明内容 4');
```

注意在列名列表中不允许省略不是自动增量的主键。

5. 列出所有非空列，并且不含自动增量的主键

列名列表必须包含所有非空列。下述语句向 test 表插入第 5 行数据，执行结果如图 4-1 第 5 行所示。

```
Insert into test (col_char, col_datetime)
    values ('字符串 5', '2024-03-25 09:08:27');
```

在这条语句中，允许空的列因为没有值，所以值为空，有默认值的列，则赋给了默认值，例如本例中的 col_tinyint 列的值为 2，虽然它是非空列，由于有了默认值，也可以省略。

下述三种类型的列可以在列名列表中省略，其他列不能省略。

- 自动增量的主键列应该省略，其值自动生成。如果没有省略，可以用 null 作为它的值。
- 允许为空的列，省略时，其值为空。
- 有默认约束的列，省略时，其值为默认值。

与之相反，不能省略的列如下所示。

- 不是自动增量的主键列。
- 没有默认约束的非空列。

6. 其他组合

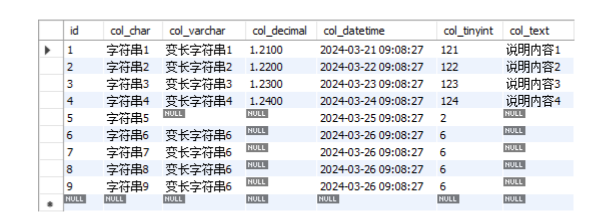
插入语句的最基本的要求是必须包含所有不能省略的列，在这个基础上再增加需要赋值的列，仅省略无需赋值的列，这是插入语句最常用的形式。下述语句向 test 表插入第 6 行数据，其中有两列是不能省略的，有两列是可以省略的，但需要为其赋值，执行结果如图 4-1 第 6 行所示。

```
Insert into test (col_char, col_varchar, col_tinyint, col_datetime)
    values ('字符串 6', '变长字符串 6', 6, '2024-03-26 09:08:27');
```

7. 一条语句插入多行数据

MySQL 还支持一条语句插入多行数据。对同一张表进行插入操作时，如果两条或多条语句的列名列表相同，可以将它们合并为一条语句，从而提高效率。下述语句向 test 表插入第 7、8、9 行数据，每一对括号提供一行数据，以逗号分隔，语句结尾用分号结束，执行结果如图 4-1 第 7、8、9 行所示。

```
Insert into test (col_char, col_varchar, col_tinyint, col_datetime) values
    ('字符串 7', '变长字符串 6', 6, '2024-03-26 09:08:27'),
    ('字符串 8', '变长字符串 6', 6, '2024-03-26 09:08:27'),
    ('字符串 9', '变长字符串 6', 6, '2024-03-26 09:08:27');
```



id	col_char	col_varchar	col_decimal	col_datetime	col_tinyint	col_text
1	字符串1	变长字符串1	1.2100	2024-03-21 09:08:27	121	说明内容1
2	字符串2	变长字符串2	1.2200	2024-03-22 09:08:27	122	说明内容2
3	字符串3	变长字符串3	1.2300	2024-03-23 09:08:27	123	说明内容3
4	字符串4	变长字符串4	1.2400	2024-03-24 09:08:27	124	说明内容4
5	字符串5	NULL	NULL	2024-03-25 09:08:27	2	NULL
6	字符串6	变长字符串6	NULL	2024-03-26 09:08:27	6	NULL
7	字符串7	变长字符串6	NULL	2024-03-26 09:08:27	6	NULL
8	字符串8	变长字符串6	NULL	2024-03-26 09:08:27	6	NULL
9	字符串9	变长字符串6	NULL	2024-03-26 09:08:27	6	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 4-1 插入语句的执行结果

4.1.3 Insert 语句与数据约束

1. 数据类型约束

每一列插入的数据必须符合该列对数据类型的要求，如有违反，插入操作失败。例如整数不能含有小数点或其他字符，字符数据的长度不能超过最大长度，汉字与英文字母一样，每个汉字按一个字符计数。日期、时间、日期时间类型的数据都有各自的格式要求，参见单元 3 的表 3-5，一些典型的出错编号及出错信息如表 4-2 所示。

表 4-2 违反数据类型约束的出错编号及出错信息举例

出错原因	出错编号及出错信息举例
字符串超出最大长度	Error Code: 1406. Data too long for column 'col_char' at row 1
不正确的实数值	Error Code: 1366. Incorrect decimal value: 'abc' for column 'col_decimal' at row 1
不正确的日期时间值	Error Code: 1292. Incorrect datetime value: '123' for column 'col_datetime' at row 1
超出可表示范围	Error Code: 1264. Out of range value for column 'col_tinyint' at row 1

2. 数据完整性约束

不能违反主键约束、外键约束、唯一性约束、非空约束和检查约束的要求，如有违反，插入操作失败，如表 4-3 所示。

表 4-3 违反数据完整性的出错编号及出错信息举例

出错原因	出错编号及出错信息举例
主键值重复	Error Code: 1062. Duplicate entry '2' for key 'publisher.PRIMARY'
外键参照错误（子表）	Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails
违反唯一性约束	Error Code: 1062. Duplicate entry 'aaa' for key 'test.col_char'
违反非空约束	Error Code: 1364. Field 'col_datetime' doesn't have a default value

对于默认约束，它只是在插入的值为空时才起作用，不会直接影响插入操作的成功或失败。

4.2 数据更新 Update

4.2.1 Update 语句

Update 语句的语法格式如下所示。

```
Update 表名 set 列名 1=值 1, 列名 2=值 2, …
    [where 条件表达式];
```

参数说明如下。

- 表名：被更新数据的表名。
- 列名：被更新数据的列名。
- 值：更新用的数据，它可以是一个值，可以是一个表达式，也可以引用当前表中任何一列的值，包括当前列的值。值的数据格式要求与插入语句完全相同。

- 条件表达式：常用的条件是“id=主键值”，参见单元 5“5.1.3 选择行 Where”一节。

4.2.2 Update 语句的使用



这里以图 4-1 所示的数据（abc 数据库的 test 表）为例，更新该表的数据。通常只需要更新某一行数据，因此需要指定条件“where id = 主键值”。

1. 更新指定列的数据

下述语句更新 id 值为 1 的那一行的 col_varchar 列的值，执行结果如图 4-2 第 1 行所示。

```
Update test set col_varchar = '第 1 行的新数据' where id = 1;
```

2. 更新多列数据

下述语句更新 id 值为 2 的那一行的 col_varchar 列和 col_text 的值，执行结果如图 4-2 第 2 行所示。

```
Update test set col_varchar = '第 2 行的新数据', col_text   =   '新的说明' where id = 2;
```

3. 更新数据时引用原有的数据

下述语句更新 id 值为 3 的那一行的 col_tinyint 列的值为原有数值减去 10，执行结果如图 4-2 第 3 行所示。

```
Update test set col_tinyint = col_tinyint - 10 where id = 3;
```

4. 更新数据时引用其他列的原有的数据

下述语句更新 id 值为 4 的那一行的 col_tinyint 列的值为原有数值减去 col_decimal 乘以 10 的结果，执行结果如图 4-2 第 4 行所示。

```
Update test set col_tinyint = col_tinyint - col_decimal * 10 where id = 4;
```

5. 更新多行数据

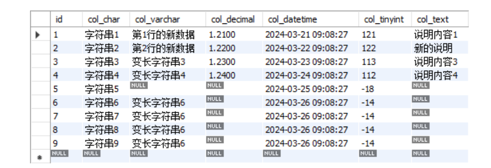
下述语句更新 id 大于等于 5 的行（共 5 行），col_tinyint 列的值为原有数值减去 20，执行结果如图 4-2 第 5、6、7、8、9 行所示。

```
Update test set col_tinyint = col_tinyint - 20 where id >= 5;
```

6. 更新所有行数据

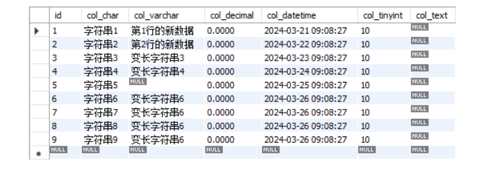
当需要更新所有行，就不需要指定条件“where id = 主键值”。下述语句更新所有行的 col_decimal 列、col_tinyint 列和 col_text 列（共 3 列），执行结果如图 4-3 所示。

```
Update test set col_decimal = 0, col_tinyint = 10, col_text = null;
```



id	col_char	col_varchar	col_decimal	col_datetime	col_tinyint	col_text
1	字符串1	第1行的新数据	1.2100	2024-03-21 09:08:27	121	说明内容1
2	字符串2	第2行的新数据	1.2200	2024-03-22 09:08:27	122	新的说明
3	字符串3	变长字符串3	1.2300	2024-03-23 09:08:27	113	说明内容3
4	字符串4	变长字符串4	1.2400	2024-03-24 09:08:27	112	说明内容4
5	字符串5	NULL	NULL	2024-03-25 09:08:27	-18	NULL
6	字符串6	变长字符串6	NULL	2024-03-26 09:08:27	-14	NULL
7	字符串7	变长字符串6	NULL	2024-03-26 09:08:27	-14	NULL
8	字符串8	变长字符串6	NULL	2024-03-26 09:08:27	-14	NULL
9	字符串9	变长字符串6	NULL	2024-03-26 09:08:27	-14	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 4-2 更新一行或多行的执行结果



id	col_char	col_varchar	col_decimal	col_datetime	col_tinyint	col_text
1	字符串1	第1行的新数据	0.0000	2024-03-21 09:08:27	10	NULL
2	字符串2	第2行的新数据	0.0000	2024-03-22 09:08:27	10	NULL
3	字符串3	变长字符串3	0.0000	2024-03-23 09:08:27	10	NULL
4	字符串4	变长字符串4	0.0000	2024-03-24 09:08:27	10	NULL
5	字符串5	NULL	0.0000	2024-03-25 09:08:27	10	NULL
6	字符串6	变长字符串6	0.0000	2024-03-26 09:08:27	10	NULL
7	字符串7	变长字符串6	0.0000	2024-03-26 09:08:27	10	NULL
8	字符串8	变长字符串6	0.0000	2024-03-26 09:08:27	10	NULL
9	字符串9	变长字符串6	0.0000	2024-03-26 09:08:27	10	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 4-3 更新所有行的执行结果



更新所有行将影响每一行的数据，并且难以恢复，需要谨慎操作。

4.2.3 Update 语句与数据约束

1. 数据类型约束

原则与数据插入时相同。

2. 数据完整性约束

不能违反主键约束、外键约束、唯一性约束、非空约束和检查约束的要求，如有违反，更新操作失败。

对于默认约束，更新操作与其完全无关，即使是更新为 null 值，默认值也不会起作用，而是直接赋

给 null 值，如果这时该列有非空约束，则会由于空值而导致更新失败。

4.3 数据删除 Delete

4.3.1 Delete 语句

Delete 语句的语法格式如下所示。

```
Delete from 表名 [where 条件表达式];
```

参数说明如下。

- 表名：被删除数据的表名。
- 条件表达式：常用的条件是“id=主键值”，参见单元 5“5.1.3 4. 模糊查询”一节。

4.3.2 Delete 语句的使用



这里以图 4-1 所示的数据（abc 数据库的 test 表）为例（数据可能更新过），删除该表的数据。

1. 删除指定行

下述语句删除 id 值为 1 的那一行。

```
Delete from test where id = 1;
```

2. 删除多行

下述语句删除 id 值大于等于 5 的行。

```
Delete from test where id >= 5;
```

3. 删除所有行

下述语句删除表中的所有行。

```
Delete from test;
```



删除所有行将删除所有数据，并且难以恢复，需要谨慎操作。

4.3.3 Delete 语句与数据约束

Delete 语句与主键约束、唯一性约束、非空约束、默认约束和检查约束无关

Delete 语句与外键约束的关系是，先删除从表中外键所在的行，然后才能删除主表中主键所在的行。

4.4 数据清空 Truncate

4.4.1 Truncate 语句

Truncate 语句的语法格式如下所示。

```
Truncate 表名;
```

参数说明如下。

- 表名：被清空数据的表名。

Truncate 语句与 Delete 语句的不同如下所示。

- Delete 语句可以删除 1 行、多行或所有数据，Truncate 语句直接无条件清空表中所有数据。
- Delete 语句不影响自增量的主键值，Truncate 语句会将自增量的主键值复位为从 1 开始。例如一张表的最大 id 是 9 时，用 delete 语句删除所有数据后，再次插入行的 id 是 10，而用 truncate 语句清空后，再次插入数据时主键值恢复为从 1 开始。

4.4.2 Truncate 语句的使用

下述语句清空表中的所有数据。

```
Truncate test;
```



清空表将无条件清空所有数据，并且不可能恢复，需要谨慎操作。

4.4.3 Truncate 语句与数据约束

Truncate 语句与主键约束、唯一性约束、非空约束、默认约束和检查约束无关

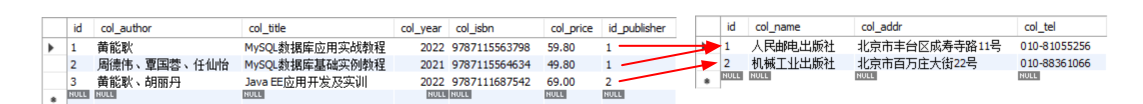
Truncate 语句与外键约束的关系是，从表可以被清空，而主表却不能被清空，试图清空主表时会提示下述出错信息，尽管参照该表的从表中已经没有数据。

```
Error Code: 1701. Cannot truncate a table referenced in a foreign key constraint (`bookinfo`.`book`, CONSTRAINT `fk_book_publisher`)
```

4.5 数据操纵与数据完整性约束

主键约束和外键约束是最重要的两种数据完整性约束，由于主键约束是强制性的，通常不会违反主键约束，本节重点对外键约束进行深入讲解。

本节以如图 4-4 所示的图书表和出版社表的数据进行讲解，创建表和初始化该数据的代码见附录 D “项目 3a 公共代码共享”，从“单元 4 数据操纵”将其中本节的公共代码复制到 MySQL Workbench 运行即可。



id	col_author	col_title	col_year	col_isbn	col_price	id_publisher
1	黄能耿	MySQL数据库应用实战教程	2022	9787115563798	59.80	1
2	周德伟、覃国蓉、任仙怡	MySQL数据库基础实例教程	2021	9787115564634	49.80	1
3	黄能耿、胡丽丹	Java EE应用开发及实训	2022	9787111687542	69.00	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL

id	col_name	col_addr	col_tel
1	人民邮电出版社	北京市丰台区成寿寺路11号	010-81055256
2	机械工业出版社	北京市百万庄大街22号	010-88361066
NULL	NULL	NULL	NULL

图 4-4 图书表和出版社表及其初始数据



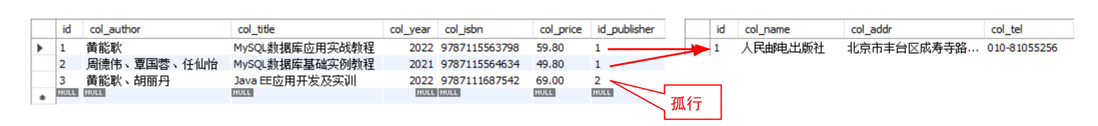
如图 4-4 所示的数据是一致的，图书表（从表）的所有外键都指向了已经存在的出版社表（主表）的主键。

1. 未创建外键约束时

1) 删除主表的行

在未创建外键约束时，如果删除主表被参照的行，例如删除 id 为 2 的机械工业出版社。

```
Delete from publisher where id = 2;
```



id	col_author	col_title	col_year	col_isbn	col_price	id_publisher
1	黄能耿	MySQL数据库应用实战教程	2022	9787115563798	59.80	1
2	周德伟、覃国蓉、任仙怡	MySQL数据库基础实例教程	2021	9787115564634	49.80	1
3	黄能耿、胡丽丹	Java EE应用开发及实训	2022	9787111687542	69.00	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL

id	col_name	col_addr	col_tel
1	人民邮电出版社	北京市丰台区成寿寺路...	010-81055256

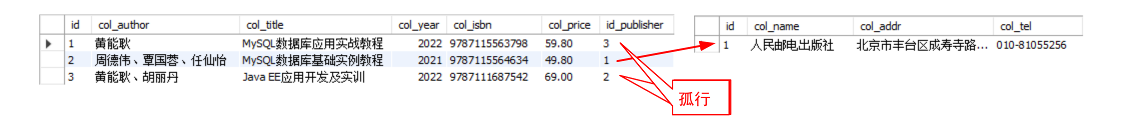
图 4-5 删除主表被参照的行引起的数据不一致

因为没有外键约束，删除操作成功执行，这时的数据出现了不一致，如图 4-5 所示。向所有出版社查询图书种数时，一共是 2 种图书，而从 book 表查询时，一共是 3 种图书，不能被出版社查询到的图书可以认为是处于“孤行”的状态。这样的数据库就有可能产生错误的数据，引起人们的不信任。

2) 更新从表的外键或添加从表的行

再看另外一种情况，更新从表的外键，使其参照不存在的出版社，代码如下。

```
Update book set id_publisher = 3 where id = 1;
```



id	col_author	col_title	col_year	col_isbn	col_price	id_publisher
1	黄能耿	MySQL数据库应用实战教程	2022	9787115563798	59.80	3
2	周德伟、覃国蓉、任仙怡	MySQL数据库基础实例教程	2021	9787115564634	49.80	1
3	黄能耿、胡丽丹	Java EE应用开发及实训	2022	9787111687542	69.00	2

id	col_name	col_addr	col_tel
1	人民邮电出版社	北京市丰台区成寿寺路...	010-81055256

图 4-6 更新从表的外键引起的数据不一致

因为没有外键约束，删除操作成功执行，这时的数据出现了不一致，如图 4-6 所示。加上前述删除主表被参照的行，数据的不一致更加严重，book 表中的 3 种图书，只有 1 种是可以被出版社查询到的，另外两行处于“孤行”状态。

2. 添加外键约束后

现在为 book 表添加外键约束，代码如下。

```
Alter table book
    add constraint fk_book_publisher foreign key (id_publisher) references publisher(id);
```

在存在数据不一致的状态下，添加外键约束是不能成功的，因此要先将数据恢复到如图 4-4 所示的状态，保证数据是在一致的状态下，才能成功执行上述添加外键约束的语句。

1) 删除主表的行

成功添加外键约束后，再删除 id 为 2 的机械工业出版社。

```
Delete from publisher where id = 2;
```

这时显示出错，提示如下错误信息。

```
Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails (`bookinfo`.`book`, CONSTRAINT `fk_book_publisher` FOREIGN KEY (`id_publisher`) REFERENCES `publisher` (`id`))
```

错误信息的意思是“不能删除或更新父表的行，外键约束失败”，这是违反了外键约束，并在括号中提供了外键约束的详细信息。

删除主表行的语句失败，数据没有改变，保持了原有的一致性。

2) 更新从表的外键或添加从表的行

现在尝试更新从表的外键，使其参照不存在的出版社，代码如下。

```
Update book set id_publisher = 3 where id = 1;
```

这时显示出错，提示如下错误信息。

```
Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`bookinfo`.`book`, CONSTRAINT `fk_book_publisher` FOREIGN KEY (`id_publisher`) REFERENCES `publisher` (`id`))
```

错误信息的意思是“不能添加或更新子表的行，外键约束失败”，这是违反了外键约束，并在括号中提供了外键约束的详细信息。

更新从表的外键的语句失败，数据没有改变，保持了原有的一致性。

3. 外键约束的作用

外键约束可以保证对数据的增删改操作不会引起由于外键参照而引起的数据不一致，违反时就会出错，增删改操作失败，保持了数据原有的一致性。

违反外键约束的出错分为两种，如表 4-4 所示。

表 4-4 违反外键约束的出错编号及出错信息

出错原因	出错的场景	出错编号及出错信息举例
对父表操作引起	删除或更新父表的行	Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails
对子表操作引起	插入或更新子表的行	Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails

在 MySQL Workbench 中显示的两种出错信息如图 4-7 所示。

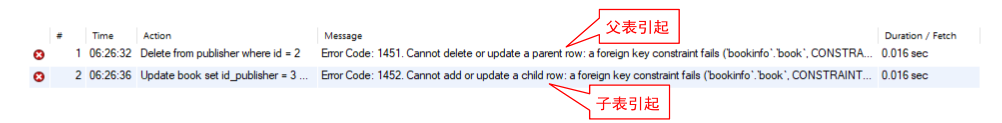


图 4-7 违反外键约束的两种出错编号及出错信息

如果没有添加外键约束，就不会出现违反外键约束的错误，增删改都可以正常进行，这并不意味着没有问题，而是隐藏了更大的问题，数据的一致性被破坏了，导致了更严重的后果，使人们对数据库产生了不信任感。



外键约束对数据库的性能有较大影响，因此有些互联网大厂严格禁止使用外键约束。外键约束可以没有，但外键却必须有，这就要求程序员在应用程序层面上实现外键约束的功能。



并不是说外键约束不重要，而是外键约束太重要了，并且还可能要由程序员来实现，因此本书在多处花了大量篇幅加以讨论。可以说，外键是全书最重要的内容之一。

【案例讲解】书店管理系统数据操纵

任务 1 数据插入——数据初始化



参考附录 D 的实战项目 2c，根据单元 2 在【案例讲解】书店管理系统需求分析部分收集到的数据（见图 2-20，更多数据在项目运行中可以看到），向出版社表、图书表、客户表、订单表和订单明细表插入数据的代码如下。

```
Insert into bs_publisher values
        (1,'人民邮电出版社','北京市丰台区成寿寺路 11 号','010-81055256'),
        (2,'机械工业出版社','北京市百万庄大街 22 号','010-88361066'),
        (3,'高等教育出版社','北京市西城区德外大街 4 号','010-58581118');
Insert into bs_book values
        (1,'黄能耿','MySQL 数据库应用实战教程',2022,'9787115563798','TP311.132.3',59.80,1),
        (2,'周德伟、覃国蓉、任仙怡','MySQL 数据库基础实例教程',2021,'9787115564634','TP311.132.3',49.80,1),
        (3,'黄能耿、胡丽丹','Java EE 应用开发及实训',2022,'9787111687542','TP312.8',69.00,2);
Insert into bs_customer values
        (1,'张三','男','13912345678','无锡市南长街 25 号'),
        (2,'李四','女','13787654321','无锡市太湖大道 1227 号'),
        (3,'门店客户','','无','自提');
Insert into bs_order values
        (1,2,'无锡市太湖大道 1227 号 8787',269.00,'已发货','2023-12-12 03:25:13','2024-02-18 22:05:00',''),
        (2,3,'自提',138.00,'已发货','2024-02-19 06:10:08','2024-02-19 06:10:55',''),
        (3,3,'自提',49.80,'已发货','2024-02-19 06:24:09','2024-02-19 06:24:55',''),
        (4,3,'自提',49.80,'已收款','2024-02-19 06:35:58','1901-01-01 00:00:00','');
Insert into bs_order_detail values (1,1,2,49.80,3,149.40),
        (2,1,1,59.80,2,119.60),
        (3,2,3,69.00,2,138.00),              -- 注意外键值是否正确地参照了主键
        (4,3,2,49.80,1,49.80),
        (5,4,2,49.80,1,49.80);
```

任务 2 项目运行——数据增删改

上述数据只是测试用的数据，项目运行过程中，将会根据业务的需要，对数据进行增删改操作。

运行附录 D 的实战项目 2c，在“图书管理”模块中，编辑一家出版社，修改一些数据，新增一本图书，删除一本图书，完成后，再从“运行日志”查看这些操作对应的 SQL 语句是什么。

【实战演练】图书借阅系统数据初始化



参考附录 D 的实战项目 2d，根据在单元 2【实战演练】图书借阅系统需求分析部分收集到的数据（见图 2-21，更多数据在项目运行中可以看到），请读者编写一些插入语句，向出版社表、图书表、图书副本表和读者表插入测试数据。

运行附录 D 的实战项目 2d，任意选择一个模块，进行编辑、新增或删除操作，完成后，再从“运行日志”查看这些操作对应的 SQL 语句是什么，想一想这些增删改语句在自行设计的项目中应该怎么写。

【单元重点】

单元小结参见本单元的【思维导图】，单元重点如下。

- 插入语句、更新句和删除语句的使用。
- 插入、更新、删除操作与数据约束（特别是主键约束和外键约束）的关系。

【课后思考】

一、选择题

1. 不是数据操纵语句的是哪一个（　　）。

   A. Create　　B. Delete　　C. Insert　　D. Update

2. 下述插入语句中，错误的是哪几个【　　　】。

   A. Insert into tab_abc (col_a, col_b) values (a, 'b');

   B. Insert into tab_abc (col_a, col_b) values (1, 'b');

   C. Insert into tab_abc (col_a, col_b) values ('a', 'b');

   D. Insert into tab_abc (col_a, col_b) values ('a', 'b', 'c');

3. 下述更新语句中，错误的是哪几个【　　　】。

   A. Update tab_abc set col_a = '1', col_b = '2';

   B. Update tab_abc sets col_a = '1', col_b = '2';

   C. Update tab_abc set col_a = '1' where col_b = '2';

   D. Update tab_abc set col_a = '1', where col_b = '2';

4. 下述删除/清空语句中，错误的是哪几个【　　　】。

   A. Delete from tab_abc;

   B. Delete from tab_abc where id = 1;

   C. Delete * from tab_abc where id = 1;

   D. Truncate from tab_abc;

二、填空题

1. 插入数据时提供了不正确的实数值，错误编号是__________。
2. 对具有外键约束的外键值进行更新时，可能会出现的错误编号是__________。

三、思考题

1. Drop 和 Delete 语句有什么区别？
2. DML 与数据完整性约束有什么关系？如果没有数据完整性约束，会出现什么现象？

【课外拓展】

1、 从网上查找并下载 MySQL8.0 参考手册（或从附录 E 的在线资源中下载），找到 Insert 一节，了解插

入语句的更多功能。

2、 从网上查找更多关于 Truncate 和 Delete 语句的区别，至少列出教材没有提到的 3 条不同之处。

单元5 数据查询

【学习目标】

知识目标

- ◆ 理解查询条件中的逻辑关系。
- ◆ 理解两张表的连接，深刻理解内连接。
- ◆ 理解查询语句中各子句的作用和出现的次序。

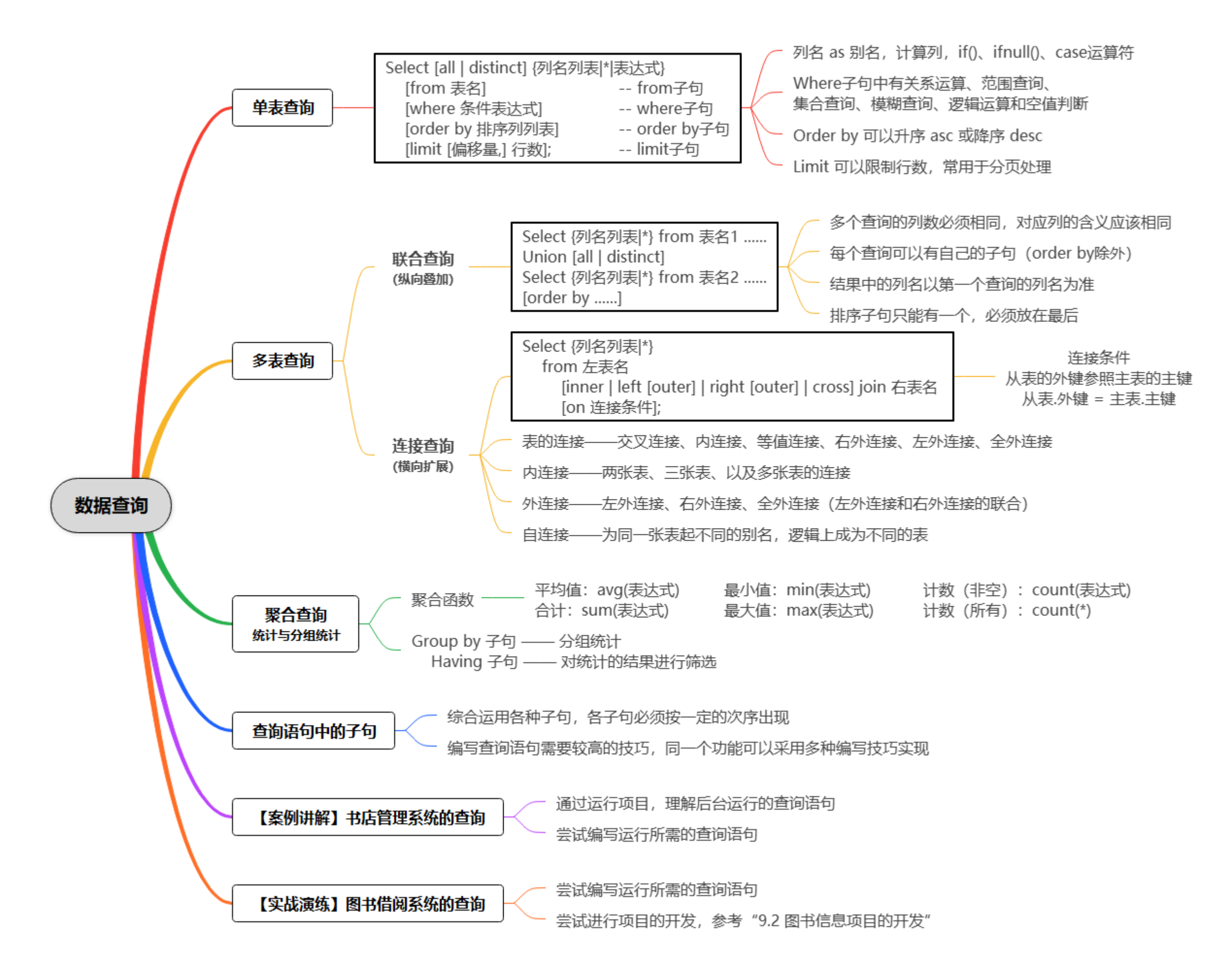
能力目标

- ◆ 熟练掌握 from、列名列表和 where 子句。
- ◆ 学会 order by 和 limit 子句。
- ◆ 学会联合查询。
- ◆ 熟练掌握内连接查询。
- ◆ 学会外连接查询。
- ◆ 掌握聚合查询（统计与分组统计）。

素质目标

- ◆ 建立万物皆有联系，联系是复杂的的观念。
- ◆ 保护用户隐私、防止数据泄露、坚守专业道德。

【思维导图】



【情景导入】

小明很快就学完了对数据的增删改操作，并且加深了对数据类型约束和数据完整性约束的理解，特别对外键约束与数据一致性的关系有了更深刻的理解。现在小明迫切地想知道如何利用这些数据，也就是数据查询，让我们同小明一起学习吧。

【知识储备】

数据查询是关系型数据库的核心功能，采用 Select 语句实现，具有十分强大的功能，在单元 1 和单元 2 中已经初步接触了这条语句，本单元全面讲解数据查询，包括单表查询、多表查询（联合查询和连接查询）以及聚合查询。

5.1 单表查询

单表查询是指对单张表的查询，这时可以查询一张表的所有数据、部分列或部分行的数据，并对查询的结果进行排序、分页等后续处理。

本节讲解的单表查询的语法格式如下所示。

```
Select [distinct] {列名列表|*|表达式}
	[from 表名]			-- from 子句
	[where 条件表达式]		-- where 子句
	[order by 排序列列表]		-- order by 子句
	[limit [偏移量,] 行数];		-- limit 子句
```

参数说明如下。

- 列名列表：将要在查询结果中列出数据的列名，以逗号分隔每一列。
- *：如果用星号代替列名列表，则表示将要列出所有列的数据。
- 表达式：表达式可以是常量、普通的表达式或函数等。
- Distinct 选项：去除结果集中的重复行（所有列都相同的行）。

子句是 Select 语句的重要组成部分，必须按语法格式中的次序出现，本节讲解下述几个子句。

1. From 子句：指定从哪张表查询，在“5.1.1 指定表 From”中讲解。
2. Where 子句：指定查询条件，选择满足条件的行，在“5.1.3 选择行 Where”中讲解。
3. Order by 子句：对结果集进行排序，在“5.1.4 排序 Order by”中讲解。

Limit 子句：限制结果集的行数，在“0



utf8mb4 字符集不是以拼音排序的，而 gbk 字符集是以拼音排序的，因此对于 utf8mb4 字符集，想要以拼音进行排序，则需要将其转换为 gbk 字符集，代码如下。

```
Select * from demo order by convert(name using gbk);
```

4. 限制行数和分页 Limit”中讲解。

本节使用如图 5-1 所示的数据进行查询操作，其中年龄是根据生日，计算 2024 年时的年龄。创建表和插入数据的代码可从附录 D“项目 3a 公共代码共享”的“单元 5 数据查询”找到，可复制使用。

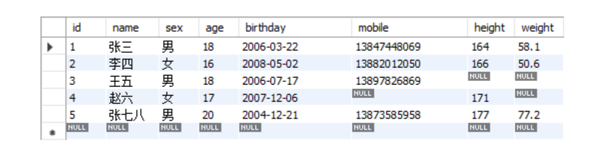
5.1.1 指定表 From

1. 指定表名

From 子句指定查询的数据来自于哪张表，格式是“from 表名”，例如下述查询语句。

```
Select *
	from demo;
```

查询结果如图 5-1 所示，显示的数据来自于 demo 表。



id	name	sex	age	birthday	mobile	height	weight
1	张三	男	18	2006-03-22	13847448069	164	58.1
2	李四	女	16	2008-05-02	13882012050	166	50.6
3	王五	男	18	2006-07-17	13897826869	NULL	NULL
4	赵六	女	17	2007-12-06	NULL	171	NULL
5	张七八	男	20	2004-12-21	13873585958	177	77.2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 5-1 demo 表的所有数据

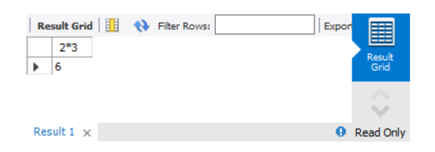


图 5-2 最简单的查询（1 行 1 列）



MySQL Workbench 查询结果中，如果右下角出现标识“Read Only”，如图 5-2 所示，表示只读，不能修改数据。否则表示可以进行增删改操作，如图 5-1 所示，这时最后一行全为 NULL。

2. 省略表名

From 子句是可以省略的，例如下述代码省略了所有子句，查询结果如图 5-2 所示。

```
Select 2*3;
```

从图 5-2 可以看到，查询的结果是 2*3 的计算结果 6。这个数据不是从表中查询得到的，而是直接计算得到的，因此还不能算是单表查询。

这条查询语句十分简单，但也是十分有用的，因为它把一个值（这个例子是表达式的运算结果）转换为一个关系，即一张表，这个例子是一张只有 1 行 1 列的表，这个查询相当于创建了一张表。



查询语句的结果永远是一个关系，即一张表，这个结果还可以作为表、集合或值，参与后续的查询，从而形成各种形式的嵌套查询，详见单元 6 的“6.1 子查询”。

5.1.2 选择列

1. 选择所有列



使用星号（*）表示选择所有列，下述语句查询 demo 表的所有数据，查询结果如图 5-1 所示。

```
Select *
    from demo;
```

2. 指定列

可以通过指定列名来选择查询的列，不再使用星号（*，它表示所有列）。

1) 指定列名（默认标题）

例如下述语句查询姓名、性别和电话三列，查询结果如图 5-3 所示。

```
Select name, sex, mobile
    from demo;
```

2) 指定列名和别名（指定标题）

还可以用 as 关键字指定列的别名（别名作为列标题显示在结果中），例如下述代码。

```
Select name as 姓名, sex as '性 别', mobile 手机号
    from demo;
```

其中的 as 关键字可以省略，如代码中 mobile 列的别名就省略了 as 关键字。别名如果含有空格或特殊字符，则需要用单引号括起来，如代码中的“性 别”中含有空格。查询结果如图 5-4 所示。

name	sex	mobile
张三	男	13847448069
李四	女	13882012050
王五	男	13897826869
赵六	女	NULL
张七八	男	13873585958

图 5-3 选择列（默认标题）

姓名	性别	手机号
张三	男	13847448069
李四	女	13882012050
王五	男	13897826869
赵六	女	NULL
张七八	男	13873585958

图 5-4 选择列（指定标题）

3. 去除重复行

当查询某个列或几个列时，在结果中可能出现完全相同的行。例如查询性别列的代码如下。

```
Select sex
	from demo;
```

结果如图 5-5 所示，如果想要消除重复行（例如想看一下有没有输入错误的性别数据），这时可以加上关键字 distinct（不同的，即没有重复的），代码如下。

```
Select distinct sex
	from demo;
```

结果如图 5-6 所示，查询的结果只有两行（如果有输入错误的性别数据，结果就会超过两行）。



sex
男
女
男
女
男

图 5-5 保留重复行的查询结果



sex
男
女

图 5-6 去除重复行的查询结果

4. 使用计算列

计算列是根据一个表达式，通过计算得到查询结果。

1) 常量

计算列可以是一个常量，例如下述语句的结果如图 5-7 所示。

```
Select '学生' 身份, name, sex, age
	from demo;
```

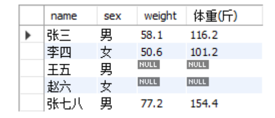
2) 表达式

计算列可以是一个表达式，通常要给这个列指定别名，例如下述语句将体重从公斤转换为市斤，别名中含有特殊符号（半角的圆括号），所以要用引号括起来，结果如图 5-8 所示。

```
Select name, sex, weight, weight*2 as '体重(斤)'
	from demo;
```

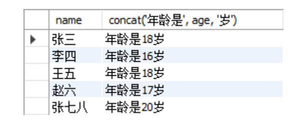
身份	name	sex	age
学生	张三	男	18
学生	李四	女	16
学生	王五	男	18
学生	赵六	女	17
学生	张七八	男	20

图 5-7 计算列（常量）



name	sex	weight	体重(斤)
张三	男	58.1	116.2
李四	女	50.6	101.2
王五	男	NULL	NULL
赵六	女	NULL	NULL
张七八	男	77.2	154.4

图 5-8 计算列（表达式）



name	concat('年龄是', age, '岁')
张三	年龄是18岁
李四	年龄是16岁
王五	年龄是18岁
赵六	年龄是17岁
张七八	年龄是20岁

图 5-9 Concat()函数

3) 函数

表达式中可以使用函数，下述语句用 concat()函数将列的值以及常量连接起来，结果如图 5-9 所示。

```
Select name, concat('年龄是', age, '岁')
	from demo;
```

函数将在单元 7 的“7.2 内置函数”讲解，常用函数参见附录 B。

5. If、ifnull 和 case

计算列中可以使用 3 个特别的关键字，它们是 if、ifnull 和 case，下面分别讲解。

1) If 函数

If 函数类似于 C/C++或 Java 语言中的三元运算符，语法格式如下。

```
If(条件表达式, 为真时的结果, 为假时的结果)
```

例如下述语句可以将以男、女表示的性别转换为“小帅哥”和“小女生”，结果如图 5-10 所示。

```
Select name 姓名,
          if(sex='男', '小帅哥', '小女生') 身份
     from demo;
```



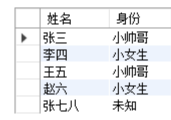
姓名	身份
张三	小帅哥
李四	小女生
王五	小帅哥
赵六	小女生
张七八	小帅哥

图 5-10 If 函数



name	手机号
张三	13847448069
李四	13882012050
王五	13897826869
赵六	缺手机号
张七八	13873585958

图 5-11 Ifnull 函数



姓名	身份
张三	小帅哥
李四	小女生
王五	小帅哥
赵六	小女生
张七八	未知

图 5-12 Case 运算符

2) Ifnull 函数

Ifnull 函数判断一个值是否为空，不为空时返回该值，为空时返回另一个值，语法格式如下。

```
Ifnull(值, 值为空时的返回值)
```

例如下述语句列出所有手机号，如果手机号为空，则显示“缺手机号”，结果如图 5-11 所示。

```
Select name,
          ifnull(mobile, '缺手机号') 手机号
     from demo;
```

3) Case 运算符

Case 运算符可以实现多个条件的判断，语法格式有两种，第一种格式如下。

```
Case 表达式
     when 等于值 1 时 then 结果 1
     when 等于值 2 时 then 结果 2
     …
     else 结果 n
end case
```

第二种格式如下。

```
Case
     when 表达式 1 为真时 then 结果 1
     when 表达式 2 为真时 then 结果 2
     …
     else 结果 n
end case
```

例如上述 if 函数的例子改写为 case 运算符，还可以有多个选择。如果将 demo 表最后一行的性别更新为 null，查询结果如图 5-12 所示。

```
Select name 姓名,
     case sex
          when '男' then '小帅哥'
          when '女' then '小女生'
          else '未知'
     end 身份
     from demo;
```

采用第二种方式时写成下述代码（具有相同的结果）。

```
Select name 姓名,
     case
          when sex='男' then '小帅哥'
          when sex='女' then '小女生'
          else '未知'
     end 身份
     from demo;
```

后一种写法更加灵活，甚至可以引用多个列的值。例如下述代码不仅根据库存量（inventory），还参

考价格（price）来决定是否需要补货，价格低于 200 元的商品，库存小于 20 件就要补货，而价格大于等于 200 元的商品，库存小于 5 件时才需要补货。

```
Select name 商品名, price 价格, inventory 库存量,          -- 这个例子没有测试数据
    case
        when price < 200 and inventory < 20 then '需补货'
        when price >= 200 and inventory < 5 then '需补货'
        else '库存充足'
    end 库存情况
    from shop_goods;
```

5.1.3 选择行 Where



Where 子句用来对数据进行筛选，选择符合查询条件的行作为查询结果。查询条件是一个条件表达式，常用的运算符如表 5-1 所示。

表 5-1 条件表达式中的常用运算符

查询类别	运算符	含义
关系运算	=、>、<、>=、<=、<>（!=）	等于、大于、小于、大等于、小等于、不等于
范围查询	between … and …、not between … and …	在…范围之间、不在…范围之间
集合查询	in (…)、not in (…)	在…集合之内、不在…集合之内
模糊查询	like、not like	类似于、不类似于
逻辑运算	and、or、not	与、或、非
空值判断	is null、is not null	为空、不为空

1. 关系表达式

关系运算符可以连接列名和常量，从而形成关系表达式，用于查询条件。最常用的是根据主键值查询指定的行，例如下述语句查询 id 为 2 的行，因为主键的唯一性，查询结果最多只有一行。

```
Select *
    from demo
    where id = 2;
```



这种查询条件在单元 3 的 Update、Delete 语句中使用过，本小节讲解的各种查询条件都可以用在 Update、Delete 语句中。

又如查询所有男性的姓名和电话，代码如下所示。

```
Select name, mobile
    from demo
    where sex = '男';
```

又如查询身高大于等于 170 厘米的数据，代码如下所示。

```
Select *
    from demo
    where height >= 170;
```

2. 范围查询

用于查询表达式的值是否在（不在）一个连续的范围内。例如查询身高在 166 到 175 厘米之间的行（含 166 和 175），代码如下所示。

```
Select *
    from demo
    where height between 166 and 175;
```

又如查询生日在 2006-01-01 与 2007-12-31 之间的人的姓名、性别和生日，代码如下所示。

```
Select name, sex, birthday
```

```
    from demo
    where birthday between '2006-01-01' and '2007-12-31';
```

3. 集合查询

用于查询表达式的值是否在（不在）一个集合中。例如查询 id 是 1 或 3 的行，由于不是连续的数字，所以应该使用集合查询，代码如下所示。

```
Select *
    from demo
    where id in (1, 3);
```

4. 模糊查询

模糊查询是非常有用的查询，这是利用通配符来达到不精确匹配的查询要求。常用的通配符有 2 种，如表 5-2 所示。

表 5-2 常用的通配符

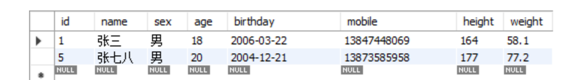
通配符	说明	实例
%	百分号，代表 0 至多个任意字符	'张%'表示以“张”起始，后接 0 至多个其他字符，即所有姓张的姓名
_	下划线，代表 1 个任意字符	'张_'表示以“张”起始，后接 1 个其他字符，即姓张的单名的姓名

例如查询所有“张”姓的行（姓名以张起头，后接任意字符），代码如下所示，查询结果如图 5-13 所示。

```
Select *
    from demo
    where name like '张%';
```

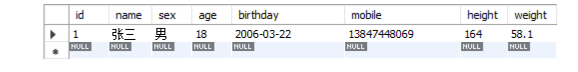
作为对比，查询所有“张”姓并且单名的行（姓名以张起头，后接一个字符），代码如下所示，查询结果如图 5-14 所示。

```
Select *
    from demo
    where name like '张_';
```



id	name	sex	age	birthday	mobile	height	weight
1	张三	男	18	2006-03-22	13847448069	164	58.1
5	张七八	男	20	2004-12-21	13873585958	177	77.2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 5-13 “张”姓的查询结果



id	name	sex	age	birthday	mobile	height	weight
1	张三	男	18	2006-03-22	13847448069	164	58.1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 5-14 “张”姓单名的查询结果

又如查询手机号码中含有数字 120 的行，代码如下所示。

```
Select *
    from demo
    where mobile like '%120%';
```

5. 逻辑表达式

需要使用多个查询条件时，可以使用 and、or 等将查询条件连接起来，形成逻辑表达式。也可以用 not 运算符，对查询条件取反。例如查询身高大于等于 170 厘米的女性，代码如下所示。

```
Select *
    from demo
    where height >= 170 and sex = '女';
```

6. 空值判断

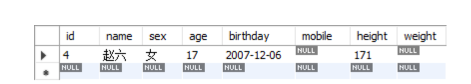
空值是没有值的，它不是 0，也不是空串（空串是长度为 0 的字符串），它表示数据的缺失。空值与 0 或空串具有不同的含义，例如某学生的考试成绩为 0 与另一学生因缺考而没有成绩是不同的。

空值判断不能使用等号（=），而是用 is null 来判断空值，用 is not null 来判断非空。

例如查询手机号为空的行，代码如下所示，查询结果如图 5-15 所示。

```
Select *
```

```
    from demo
    where mobile is null;
```



id	name	sex	age	birthday	mobile	height	weight
4	赵六	女	17	2007-12-06	NULL	171	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 5-15 手机号为空的行

id	name	sex	age	birthday	mobile	height	weight
1	张三	男	18	2006-03-22	13847448069	164	58.1
5	张七八	男	20	2004-12-21	13873585958	177	77.2
2	李四	女	16	2008-05-02	13882012050	166	50.6
3	王五	男	18	2006-07-17	13897826869	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 5-16 手机号非空的行

查询手机号非空的行，代码如下所示，查询结果如图 5-16 所示。

```
Select *
    from demo
    where mobile is not null;
```

5.1.4 排序 Order by



单元 2 讲过关系的 6 项基本特征（参见表 2-4），其中两项基本特征是“行的次序无关性”，以及“列的次序无关性”，查询的结果也是一个关系，也适用这 6 项基本特征，因此查询的结果是无序的。

可以通过指定列名列表来指定查询结果中列的次序，而指定行的次序则要通过 order by 子句来实现，这样才能以合适的排序方式将结果呈现给用户，例如按生日或按身高排序输出，方便用户的阅读。

1. 升序排序

升序排序是将数据从小到大进行排序，用关键字 asc 表示，这是默认的排序方式，因此可以省略 asc 关键字。例如按生日进行升序排序，代码如下所示，查询结果如图 5-17 所示。

```
Select *
    from demo
    order by birthday asc;
```

2. 降序排序

降序排序是从大到小进行排序，用关键字 desc 表示。例如按生日进行降序排序，代码如下所示，查询结果如图 5-18 所示。

```
Select *
    from demo
    order by birthday desc;
```

id	name	sex	age	birthday	mobile	height	weight
5	张七八	男	20	2004-12-21	13873585958	177	77.2
1	张三	男	18	2006-03-22	13847448069	164	58.1
3	王五	男	18	2006-07-17	13897826869	NULL	NULL
4	赵六	女	17	2007-12-06	NULL	171	NULL
2	李四	女	16	2008-05-02	13882012050	166	50.6
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 5-17 生日列的升序排序

id	name	sex	age	birthday	mobile	height	weight
2	李四	女	16	2008-05-02	13882012050	166	50.6
4	赵六	女	17	2007-12-06	NULL	171	NULL
3	王五	男	18	2006-07-17	13897826869	NULL	NULL
1	张三	男	18	2006-03-22	13847448069	164	58.1
5	张七八	男	20	2004-12-21	13873585958	177	77.2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 5-18 生日列的降序排序

3. 多个排序列

前述的例子只有一个排序列，仅对生日进行排序。也可以指定多个排序列，这时先以第一排序列进行排序，如果第一排序列的排序相同，再按第二排序列进行排序，以此类推。

例如对性别列和生日列进行排序，即先按性别进行排序，然后在相同的性别中，再按生日进行排序，代码如下所示，查询结果如图 5-19 所示。

```
Select *
    from demo
    order by sex, birthday;
```

将上述代码修改一下，将生日列改为降序，其余不变，代码如下所示，查询结果如图 5-20 所示。

```
Select *
```

```
    from demo
    order by sex, birthday desc;
```

id	name	sex	age	birthday	mobile	height	weight
4	赵六	女	17	2007-12-06	NULL	171	NULL
2	李四	女	16	2008-05-02	13882012050	166	50.6
5	张七八	男	20	2004-12-21	13873585958	177	77.2
1	张三	男	18	2006-03-22	13847448069	164	58.1
3	王五	男	18	2006-07-17	13897826869	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 5-19 性别（升序）和生日（升序）排序

id	name	sex	age	birthday	mobile	height	weight
2	李四	女	16	2008-05-02	13882012050	166	50.6
4	赵六	女	17	2007-12-06	NULL	171	NULL
3	王五	男	18	2006-07-17	13897826869	NULL	NULL
1	张三	男	18	2006-03-22	13847448069	164	58.1
5	张七八	男	20	2004-12-21	13873585958	177	77.2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 5-20 性别（升序）和生日（降序）排序



utf8mb4 字符集不是以拼音排序的，而 gbk 字符集是以拼音排序的，因此对于 utf8mb4 字符集，想要以拼音进行排序，则需要将其转换为 gbk 字符集，代码如下。

```
Select * from demo order by convert(name using gbk);
```

5.1.5 限制行数和分页 Limit

1. 限制行数

Jitor 实训 | 附录 C 实训 5-4

如果表中的数据太多，例如达到几百上千行，甚至上百万行，这时显示所有行将是不现实的，可以指定只显示其中的部分行，使用 limit 子句来实现。例如下述代码。

```
Select *
    from demo
    limit 3;
```

运行的结果是只显示前 3 行。

2. 分页

Limit 关键字通常用于分页，语法格式如下。

```
Select {*|列名列表}
    from 表名
    limit (页号-1)*每页行数, 每页行数;
```

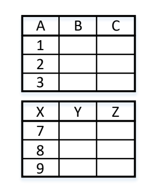
当前 demo 表中共有 5 行数据，如果把每页行数设定为 2，将会显示为 3 页。显示这 3 页的代码如下。

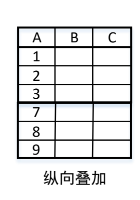
```
Select * from demo limit 0, 2;          -- 显示第 1 页，每页 2 行，0 = (1-1)*2
Select * from demo limit 2, 2;          -- 显示第 2 页，每页 2 行，2 = (2-1)*2
Select * from demo limit 4, 2;          -- 显示第 3 页，每页 2 行，4 = (3-1)*2，这是最后一页，结果只有 1 行
```

5.2 联合查询 Union

上一节讲解了对单表的查询，本节和下一节要讲解对多表的查询，就是说，从多张表的数据进行查询，最后得到一个结果集（一张表），多表查询又分为下述两种情况，如图 5-21 所示。

- 纵向叠加：将多张表纵向叠加在一起，成为一张表，这种情况要求列的数量相同，对应列的含义应该相同。这是本节讲解的联合查询。
- 横向扩展：将多张表横向扩展开来，成为一张表，这通常要求表与表之间要有联系，即外键参照主键的联系。将在下一节的连接查询中讲解。





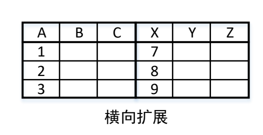


图 5-21 多表查询的处理

5.2.1 联合查询的语法格式

联合查询的语法格式如下所示。

```
Select {列名列表|*} from 表名 1
      [where 条件表达式]
Union [all | distinct]
Select {列名列表|*} from 表名 2
      [where 条件表达式]
Union [all | distinct]
Select {列名列表|*} from 表名 n
      [where 条件表达式]
[order by];
```

参数说明如下。

- Select 语句：不含排序子句的 Select 语句。
- Union：可以用 Union 联合两个到任意多个查询。
- All 选项：保留结果集中的重复行。
- Distinct 选项：去除结果集中的重复行，所有行都是不同的，这是默认选项。

联合查询的要求如下。

- 两个或多个查询的列数相同，如果不相同，则会出错。
- 两个或多个查询对应列的含义相同，如果不相同，则会造成误解。
- 结果中的列名以第一个查询的列名为准，第二个或之后的查询的列名不起任何作用。
- 每个查询可以有各自的查询条件等子句。
- 排序子句只能有一个，并且是针对整个联合的，因此必须放在最后。排序列名使用第一个查询的列名。

5.2.2 联合查询的使用

1. 联合查询两张表



本节使用图 5-22 所示 demo 表和 demo1 表的数据演示两张表的联合查询，创建表和初始化该数据的代码见附录 D“项目 3a 公共代码共享”的“单元 5 数据查询”。

id	name	sex	age	birthday	mobile	height	weight
1	张三	男	18	2006-03-22	13847448069	164	58.1
2	李四	女	16	2008-05-02	13882012050	166	50.6
3	王五	男	18	2006-07-17	13897826869	NULL	NULL
4	赵六	女	17	2007-12-06	NULL	171	NULL
5	张七八	男	20	2004-12-21	13873585958	177	77.2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

id	col_name	col_sex	col_age	col_mobile	col_tel
1	张老师	男	38	13816033758	0510-37585637
2	李老师	女	35	13895468568	0510-37585638
3	王老师	女	42	13877508776	0510-37585638
NULL	NULL	NULL	NULL	NULL	NULL

图 5-22 demo 表和 demo1 表的数据

从图 5-22 中看到，demo 表和 demo1 表的列数量和列名都是不同的，但是列的含义基本相同，因此可以将这两张表联合起来。

例如选择两张表中共同的列，将它们联合起来，代码如下所示，结果如图 5-23 所示。

```
Select id, name, sex, age, mobile
      from demo
Union
Select id, col_name, col_sex, col_age, col_mobile
      from demo1;
```

这两个 Select 语句通过 Union 关键字联合起来，形成一条语句，因此只在最后一个 Select 语句后加上分号，作为整条语句的结束。

id	name	sex	age	mobile
1	张三	男	18	13847448069
2	李四	女	16	13882012050
3	王五	男	18	13897826869
4	赵六	女	17	NULL
5	张七八	男	20	13873585958
1	张老师	男	38	13816033758
2	李老师	女	35	13895468568
3	王老师	女	42	13877508776

图 5-23 两张表的联合（共同列）

id	name	sex	age	mobile	固定电话	height
1	张三	男	18	13847448069	无	164
2	李四	女	16	13882012050	无	166
3	王五	男	18	13897826869	无	NULL
4	赵六	女	17	NULL	无	171
5	张七八	男	20	13873585958	无	177
1	张老师	男	38	13816033758	0510-37585637	NULL
2	李老师	女	35	13895468568	0510-37585638	NULL
3	王老师	女	42	13877508776	0510-37585638	NULL

图 5-24 两张表的联合（含非共同列）

还可以选择更多的列，如果另一张表没有对应的列，则用常量或 null 替代，代码如下所示，结果如图 5-24 所示。

```
Select id, name, sex, age, mobile, '无' 固定电话, height
	from demo
Union
Select id, col_name, col_sex, col_age, col_mobile, col_tel, null
	from demo1;
```

2. 联合查询中的子句

如上所述，每个查询可以有各自的查询条件等子句，而排序子句只能有一个，并且必须放在最后。

例如下述代码的第一个查询有 where 子句，排除没有手机号的人员，排序是对总的结果进行的，但是排序的列名要用第一个查询的列名，而不能用第二个查询的列名。

```
Select id, name, sex, age, mobile, '无' 固定电话, height
	from demo
	where mobile is not null		-- 有自己的 where 子句
Union
Select id, col_name, col_sex, col_age, col_mobile, col_tel, null
	from demo1
	where col_mobile is not null		-- 有自己的 where 子句
order by sex, age;			-- order by 子句只能有一个，并且必须放在最后
```

3. 联合查询同一张表

联合查询不必是对两张不同的表进行，也可以对同一张表分别进行两次查询，再将结果联合起来。例如下述语句，查询结果如图 5-25 所示。

```
Select id, name, sex, age, mobile, height
	from demo
	where sex = '男'
Union
Select id, name, sex, age, mobile, height
	from demo
	where height > 170;
```

上述这条语句等价于下述语句。

```
Select id, name, sex, age, mobile, height
	from demo
	where sex = '男' or height > 170;
```

4. 保留结果集中的重复行

Union 的默认选项是 distinct（去除结果集中的重复行），如果想要保留结果集中的重复行，则需要明确指定 all 选项。将上述代码改写一下，加上 all 选择，代码如下所示，查询结果如图 5-26 所示。

```
Select id, name, sex, age, mobile, height
	from demo
	where sex = '男'
Union all
Select id, name, sex, age, mobile, height
	from demo
	where height > 170;
```

id	name	sex	age	mobile	height
1	张三	男	18	13847448069	164
3	王五	男	18	13897826869	NULL
5	张七八	男	20	13873585958	177
4	赵六	女	17	NULL	171

图 5-25 同一张表两个查询的联合

id	name	sex	age	mobile	height
1	张三	男	18	13847448069	164
3	王五	男	18	13897826869	NULL
5	张七八	男	20	13873585958	177
4	赵六	女	17	NULL	171
5	张七八	男	20	13873585958	177

图 5-26 保留结果集中重复行的联合

从图 5-26 中可以看到，id 为 5 的行重复出现在结果集中，这是因为这一行既满足第一个查询，也满足第二个查询，在指定 all 选项后，重复的行就保留在结果集中了。

5.3 连接查询 Join

上一节讲过多表查询有两种情况（参见图 5-21），第一种是纵向叠加，即联合查询，在上一节讲解，第二种是横向扩展，即连接查询，在本节讲解。

连接查询的实质是在设计关系数据库时，对实体进行拆分的逆过程，将拆分出来的表根据查询的目的重新连接起来。因为拆分出来的表都是原子性的，连接的时候就可以非常灵活，以满足现实世界的各种需求，这是关系数据库成为主流技术的重要原因之一。

连接查询的语法格式如下所示。

```
Select {列名列表|*}
    from 左表名
        [inner | left [outer] | right [outer] | cross] join 右表名
        [on 连接条件];
```

参数说明如下。

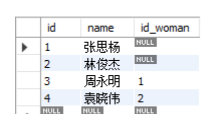
- 左表名：join 关键字左侧的表的表名。
- 右表名：join 关键字右侧的表的表名。
- Inner 选项：指定连接的类型是内连接，这是默认值，可以省略。
- Left outer 选项：指定连接的类型是左外连接（列出左表名的所有行），可以省略 outer 关键字。
- Right outer 选项：指定连接的类型是右外连接（列出右表名的所有行），可以省略 outer 关键字。
- Cross 选项：指定连接的类型是交叉连接，交叉连接不需要“on 连接条件”。
- 连接条件：对于交叉连接之外的连接查询，需要指定连接条件，详见下面例子的讲解。



MySQL 的 Select 语句从语法上不提供全外连接，而是通过左外连接和右外连接的联合查询来实现全外连接。

5.3.1 两张表的连接

本小节以如图 5-27 所示的男人表（man）和女人表（woman）讲解两张表的连接，创建表和初始化该数据的代码见附录 D“项目 3a 公共代码共享”的“单元 5 数据查询”。



id	name	id_woman
1	张思杨	NULL
2	林俊杰	NULL
3	周永明	1
4	袁晓伟	2
NULL	NULL	NULL



id	name
1	李晓燕
2	陈华
3	王婧雅
NULL	NULL

图 5-27 男人表（man）和女人表（woman）的数据

图 5-27 所示的男人表（man）的外键 id_woman 参照女人表（woman）的主键 id，这个一对一联系代表两人之间的夫妻关系。

1. 交叉连接



交叉连接列出两张表的行的所有组合，而不考虑其他任何条件。例如下述代码列出男人和女人的所有组合，不考虑是否已婚和婚姻关系。

```
Select *
    from man cross join woman;        -- 不需要连接条件
```

结果如图 5-28 所示，这种情况在婚介所里对未婚男女进行配对筛选时会用到。

交叉连接运行结果的行数是两张表行数的乘积，例如图 5-28 的行数是 3*4 = 12 行。对于行数较多的两张表，交叉连接结果的行数就会变得相当大。

交叉连接在实际项目中极少用到，但交叉连接有助于理解其他连接。后面要讲的内连接和外连接都是交叉连接的特例，是满足一定条件的交叉连接的子集。

2. 内连接

内连接是在交叉连接的基础上，加上一个条件。在这个例子中，这个条件是如下的婚姻关系。

man 的 id_woman（等于）woman 的 id

这个连接条件体现了外键参照的如下原则。

从表的外键（参照）主表的主键

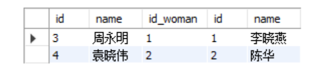
写成连接查询的代码，如下所示，on 子句是连接条件。

```
Select *
    from man inner join woman
        on man.id_woman = woman.id;
```

查询结果如图 5-29 所示，结果显示的是两对夫妻，就是从交叉连接的结果中选择符合连接条件的结果，即图 5-28 中加亮显示的两对夫妻。

id	name	id_woman	id	name
1	张思杨	NULL	3	王婧雅
1	张思杨	NULL	2	陈华
1	张思杨	NULL	1	李晓燕
2	林俊杰	NULL	3	王婧雅
2	林俊杰	NULL	2	陈华
2	林俊杰	NULL	1	李晓燕
3	周永明	1	3	王婧雅
3	周永明	1	2	陈华
3	周永明	1	1	李晓燕
4	袁晓伟	2	3	王婧雅
4	袁晓伟	2	2	陈华
4	袁晓伟	2	1	李晓燕

图 5-28 交叉连接的结果



id	name	id_woman	id	name
3	周永明	1	1	李晓燕
4	袁晓伟	2	2	陈华

图 5-29 内连接和等值连接

从逻辑上看，是将从表连接到主表，连接的条件是从表的外键等于主表的主键，这个条件与外键的定义是一致的。



内连接是最常用的一种连接，因此 inner join 中的 inner 可以省略，不加修饰的关键字 join 本身就表示是内连接。

3. 等值连接

等值连接是与内连接等价的一种写法，读者可以自由选择等值连接或内连接实现相同的功能。等值连接的语法格式如下所示。

```
Select {列名列表|*}
    from 表名 1, 表名 2
    where 表名 1.列名 1 = 表名 2.列名 2
```

参数说明如下。

- 表名 1，表名 2：需要连接的两张。
- 表名 1.列名 1 = 表名 2.列名 2：连接条件，即“从表的外键=主表的主键”。

例如前述的内连接查询语句。

```
Select *
	from man inner join woman
		on man.id_woman = woman.id;
```

改写为等值连接查询，代码如下，查询结果完全相同，如图 5-29 所示。

```
Select *
	from man, woman
	where man.id_woman = woman.id;
```

从语法格式上看，将内连接的 inner join 改为逗号，把 on 改为 where，连接的条件不变，但是写在 where 子句中，就成为了等值连接。

4. 右外连接

图 5-29 只显示了已婚的人，未婚的人并没有显示出来。如果还要显示未婚的女人，因为 woman 表在 join 的右边，就要用右外连接，把右边的人全部显示出来。代码如下，运行结果如图 5-30 所示。

```
Select *
	from man right outer join woman
		on man.id_woman = woman.id;
```

上述代码在内连接的结果上，还会列出右边表中的所有行。查询语句中的 outer 可以省略，但关键字 right 不能省略。

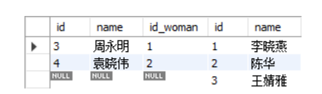


图 5-30 右外连接

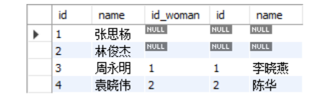


图 5-31 左外连接

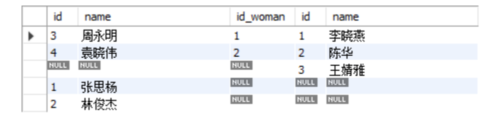


图 5-32 全外连接

5. 左外连接

如果要显示未婚的男人，因为 man 表在 join 的左边，就要用左外连接，把左边的人全部显示出来。代码如下，运行结果如图 5-31 所示。

```
Select *
	from man left outer join woman
		on man.id_woman = woman.id;
```

上述代码在内连接的结果上，还会列出左边表中的所有行。



左外连接和右外连接是互为镜像的，如果将 Join 两边的表互换，同时关键字 right 和 left 互换，查询的结果是相同的。

6. 全外连接

有时还会有这个需求，就是把左右两边的人全部都显示出来。在 SQL Server 中，可以用下述语句实现（MySQL 不支持）。

```
Select *
	from man full outer join woman
		on man.id_woman = woman.id;		-- MySQL 不支持 full outer join
```

其中 full outer join 是全外连接的意思，但是 MySQL 不支持这个语法，而是采用将左外连接和右外连接的联合（Union）来实现，代码如下。

```
Select *
	from man right outer join woman
		on man.id_woman = woman.id
union
Select *
```

```
from man left outer join woman
        on man.id_woman = woman.id;
```

其中 union 是联合查询，运行结果如图 5-32 所示。

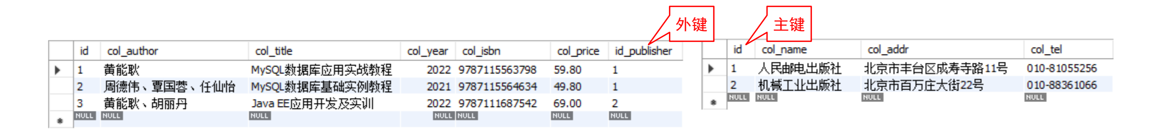
5.3.2 内连接查询



上一小节讲解了两张表的各种连接，本小节用实例来讲解其中最重要的一种，内连接查询。

1. 两张表的内连接

首先回顾一下单元 2 “2.2.2 图书信息数据库的设计”的双表设计方案，将图书信息拆分为两张表：出版社表和图书表，参见图 2-2 和图 2-3，这两张表的数据输入数据库后，如图 5-33 所示。



id	col_author	col_title	col_year	col_isbn	col_price	id_publisher
1	黄能耿	MySQL数据库应用实战教程	2022	9787115563798	59.80	1
2	周德伟、覃国蓉、任仙怡	MySQL数据库基础实例教程	2021	9787115564634	49.80	1
3	黄能耿、胡丽丹	Java EE应用开发及实训	2022	9787111687542	69.00	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL

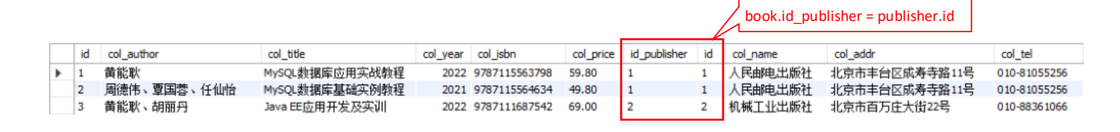
id	col_name	col_addr	col_tel
1	人民邮电出版社	北京市丰台区成寿寺路11号	010-81055256
2	机械工业出版社	北京市百万庄大街22号	010-88361066
NULL	NULL	NULL	NULL

图 5-33 出版社表（publisher）和图书表（book）的数据

下述语句将图书表（book）和出版社表（publisher）连接起来，查询结果如图 5-34 所示。

```
Use bookinfo;            -- 创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”

Select *
        from book inner join publisher
                on book.id_publisher = publisher.id;
```



id	col_author	col_title	col_year	col_isbn	col_price	id_publisher	id	col_name	col_addr	col_tel
1	黄能耿	MySQL数据库应用实战教程	2022	9787115563798	59.80	1	1	人民邮电出版社	北京市丰台区成寿寺路11号	010-81055256
2	周德伟、覃国蓉、任仙怡	MySQL数据库基础实例教程	2021	9787115564634	49.80	1	1	人民邮电出版社	北京市丰台区成寿寺路11号	010-81055256
3	黄能耿、胡丽丹	Java EE应用开发及实训	2022	9787111687542	69.00	2	2	机械工业出版社	北京市百万庄大街22号	010-88361066

图 5-34 内连接查询结果

查询结果中两张表横向排列，核心是如下所示的连接条件。

book 的 id_publisher（等于）publisher 的 id

这个连接条件体现了外键参照的如下原则。

从表的外键（参照）主表的主键

由于图书表第 1、2 行的外键值都是 1，因此出版社的信息重复出现了，这种重复出现的数据在设计数据库时是要尽力避免的，而在查询时则是必需的，通过连接查询把重复的数据展现给用户。



内连接是关系数据库的核心技术，也是主键和外键概念的延伸，一定要熟练掌握，深刻理解。

在实际开发中，通常不需要列出所有列，特别是不需要列出连接条件相关的列。因此上述代码通常会改写为如下所示的代码。

```
Select book.id, col_author, col_title, col_year, col_isbn, col_price, col_name, col_addr, col_tel
        from book inner join publisher
                on book.id_publisher = publisher.id;
```

在不同的表中可能会有相同名称的列名（列名相同而表名不同），为了区分它们，需要指定列名所属的表。例如上述两张表的主键都是 id，因此必须在列名前指定正确的表名，并用小数点分隔，例如写为

book.id，才能正确地区分它们。

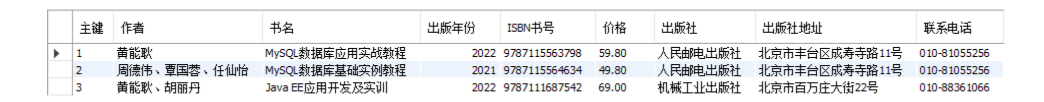


没有歧义的列的前面可以省略表名，通常互联网大厂的编码规范要求不能省略表名，以避免项目维护过程中可能出现的 bug。

规范的写法是对所有列都要指定所属的表名，这样可以更加准确，没有歧义。对于连接的条件“外键=主键”，更应该指定外键或主键所属的表名。因此上述语句的规范写法如下所示，重点显示每列所属的表名，表名和列名之间用小数点分隔。

```
Select book.id 主键,
            book.col_author 作者,
            book.col_title 书名,
            book.col_year 出版年份,
            book.col_isbn ISBN 书号,
            book.col_price 价格,
            publisher.col_name 出版社,
            publisher.col_addr 出版社地址,
            publisher.col_tel 联系电话
      from book inner join publisher
            on book.id_publisher = publisher.id;
```

运行结果如图 5-35 所示，与单元 2 的图 2-1 进行对照，可以看到图书信息数据是完全相同的。



主键	作者	书名	出版年份	ISBN书号	价格	出版社	出版社地址	联系电话
1	黄能耿	MySQL数据库应用实战教程	2022	9787115563798	59.80	人民邮电出版社	北京市丰台区成寿寺路11号	010-81055256
2	周德伟、覃国蓉、任仙怡	MySQL数据库基础实例教程	2021	9787115564634	49.80	人民邮电出版社	北京市丰台区成寿寺路11号	010-81055256
3	黄能耿、胡丽丹	Java EE应用开发及实训	2022	9787111687542	69.00	机械工业出版社	北京市百万庄大街22号	010-88361066

图 5-35 连接查询得到的图书信息数据

2. 三张表的内连接

三张表的内连接就是先将两张表连接好，再同第三张表进行一次内连接。因此，在理解了两张表连接的基础上，就很容易理解三张表的连接。

这里以单元 3“【案例讲解】创建书店管理数据库”为例加以讲解，它的 EER 图如图 5-36 所示，创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”。

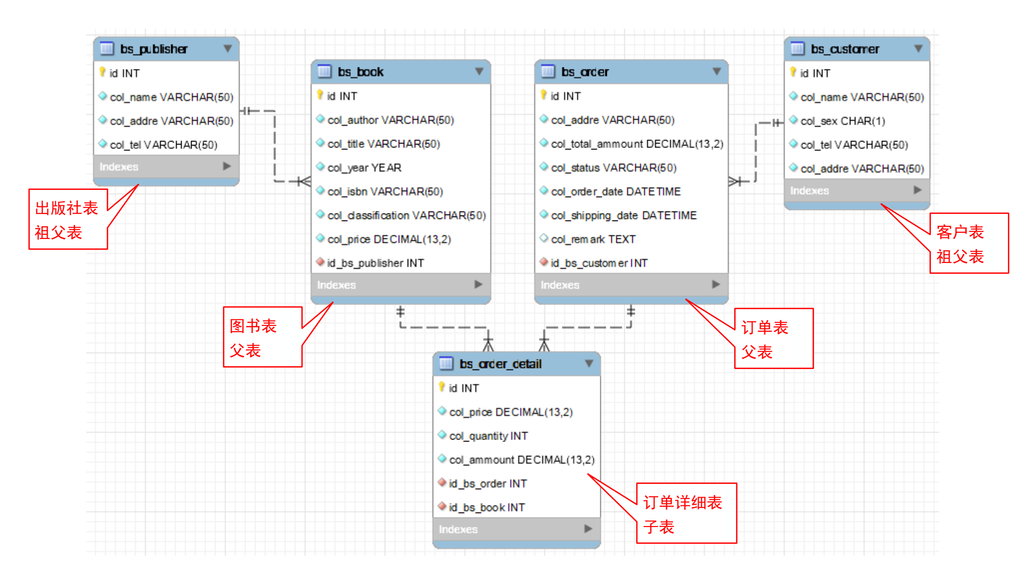


图 5-36 书店管理数据库的 EER 图

从图 5-36 选取有联系的三张表，可以分为两种不同的情况，如下所示。

● 子表有两张父表：在图 5-36 的例子中，选取订单明细表（bs_order_detail）、订单表（bs_order）和图书表（bs_book），这时订单明细表有两张父表。

● 子表有一张父表，父表再有祖父表：在图 5-36 的例子中，选取订单明细表、订单表和客户表（bs_customer），这时订单明细表是子表，订单表是父表，客户表是祖父表。

虽然从逻辑上来看，三张表的联系有不同的情况，但连接查询的写法是没有区别的。订单明细表（bs_order_detail）、订单表（bs_order）和图书表（bs_book）的内连接的语句如下。

```
Use bookstore;          -- 创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”
Select *                -- 可以根据需要选择合适的列
    from bs_order_detail
        join bs_order on bs_order_detail.id_bs_order = bs_order.id       -- 订单明细表连接到订单表
        join bs_book on bs_order_detail.id_bs_book = bs_book.id;         -- 订单明细表连接到图书表
```

上述代码中的连接关系是，从表(bs_order_detail)连接到主表(bs_order)，同时再另一个主表(bs_book)。

订单明细表（bs_order_detail）、订单表（bs_order）和客户表（bs_customer）的内连接的语句如下。

```
Select *                -- 可以根据需要选择合适的列
    from bs_order_detail
        join bs_order on bs_order_detail.id_bs_order = bs_order.id             -- 订单明细表连接到订单表
        join bs_customer on bs_order.id_bs_customer = bs_customer.id;          -- 订单表再连接到客户表
```

上述代码中的连接关系是，从表（bs_order_detail）连接到主表（bs_order），然后再连接到上一层的主表（祖父表，bs_customer）。

3. 多张表的内连接

SQL 支持任意多张表的连接，n 张表连接查询的语法格式如下。

```
Select  列名列表
    from  表名 1
        join  表名 2 on  表名 1.列名  =  表名 2.列名
        join …… on ……
        join  表名 n on  表名 m.列名  =  表名 n.列名;
```

代码中共有 n-1 个 join…on…，参数说明如下。

● 列名：主键或外键，等于号的一边是从表的外键，另一边是主表的主键。

● 表名：是表名 1 到表名 n 中的某个表名。

对于如图 5-36 所示的书店管理数据库，一共有 5 张表，这 5 张表的内连接代码如下如示。

```
Select *                -- 可以根据需要选择合适的列
    from bs_order_detail
        join bs_order on bs_order_detail.id_bs_order = bs_order.id             -- 订单明细表连接到订单表
        join bs_customer on bs_order.id_bs_customer = bs_customer.id           -- 订单表再连接到客户表
        join bs_book on bs_order_detail.id_bs_book = bs_book.id                -- 订单明细表连接到图书表
        join bs_publisher on bs_book.id_bs_publisher = bs_publisher.id;        -- 图书表再连接到出版社表
```



有直接联系的两张表才能连接，没有直接联系的两张只能通过连接的路径一级一级地连接。例如从图书表不能直接连接到客户表，而必须先连接到订单表，再连接到客户表。

5.3.3 外连接查询



附录 C 实训 5-8

在图 5-33 所示的数据中，如果出版社表新增了一家名为“新的出版社”的出版社，代码如下。

```
Use bookinfo;

Insert into publisher values (null,'新的出版社','xxx','nnn');
```

因为新的出版社还没有发行图书，因此用内连接查询将查询不到这家出版社。这时要用外连接才可

以查询到还没有发行图书的新的出版社，代码如下。

```
Select *
	from book right join publisher
		on book.id_publisher = publisher.id;
```

因为出版社表位于 join 的右侧，所以用右外连接查询，结果如图 5-37 所示。

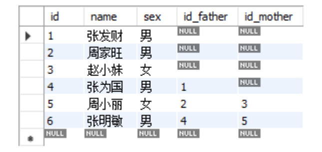
id	col_author	col_title	col_year	col_isbn	col_price	id_publisher	id	col_name	col_addr	col_tel
2	周德伟、覃国蓉、任仙怡	MySQL数据库基础实例教程	2021	9787115564634	49.80	1	1	人民邮电出版社	北京市丰台区成寿寺路11号	010-81055256
1	黄能耿	MySQL数据库应用实战教程	2022	9787115563798	59.80	1	1	人民邮电出版社	北京市丰台区成寿寺路11号	010-81055256
3	黄能耿、胡丽丹	Java EE应用开发及实训	2022	9787111687542	69.00	2	2	机械工业出版社	北京市百万庄大街22号	010-88361066
NULL	NULL	NULL	NULL	NULL	NULL	NULL	3	新的出版社	xxx	nnn

图 5-37 右外连接列出所有出版社

5.3.4 自连接查询



前面讲解的连接是两张表或多张表之间的连接，在有些情况下，会出现一张表与自身进行连接的需求。为此用一张家庭成员表（family）作为演示，数据如图 5-38 所示，家庭成员关系如图 5-39 所示，演示自连接所形成的家庭关系，创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”。



id	name	sex	id_father	id_mother
1	张发财	男	NULL	NULL
2	周家旺	男	NULL	NULL
3	赵小妹	女	NULL	NULL
4	张为国	男	1	NULL
5	周小丽	女	2	3
6	张明敏	男	4	5
NULL	NULL	NULL	NULL	NULL

图 5-38 初始数据

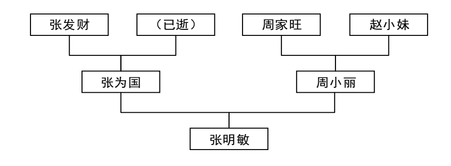


图 5-39 家庭成员关系

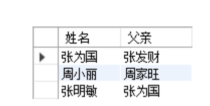
家庭成员表（family）有两个外键：id_father 参照父亲 id，id_mother 参照母亲 id，这两个外键都连接到自身，如果外键为空，表示父亲或母亲已去世。

1. 查询每个人的父亲

首先查询每个人的父亲，如果父亲不在世则无需列出。

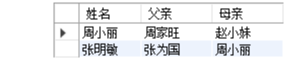
```
Select me.name  姓名,
		father.name  父亲
	from family as me
		join family as father on me.id_father=father.id;
```

其中的 from family join family 就是自己连接自己，这时一个最大问题是如何分辨谁是谁，解决的办法是指定一个别名，其中一个是我（me），另一个是父亲（father），这样就把这张表在逻辑上看成是两张表（“我”和“父亲”两张表），把这两张逻辑上的表连接起来，我的 name 就是我的姓名，父亲的 name 就是父亲的姓名。运行结果如图 5-40 所示。



姓名	父亲
张为国	张发财
周小丽	周家旺
张明敏	张为国

图 5-40 查询每个人的父亲



姓名	父亲	母亲
周小丽	周家旺	赵小妹
张明敏	张为国	周小丽

图 5-41 查询每个人的父亲和母亲

姓名	性别	父亲	母亲
张发财	男	NULL	NULL
周家旺	男	NULL	NULL
赵小妹	女	NULL	NULL
张为国	男	张发财	NULL
周小丽	女	周家旺	赵小妹
张明敏	男	张为国	周小丽

图 5-42 查询每个人的所有信息

自连接的实质是在一张表的两个虚拟副本之间的连接，表的虚拟副本就是表的一个别名，在物理上是同一张表，拥有相同的结构和数据，逻辑上作为不同的表处理。在上述例子中，family 表有两个虚拟副本，一个是 me，另一个是 father，这两个名字的表在物理上是同一张表，在逻辑上是两张表，各自的逻辑用途不同，分别表示自己和父亲，这两张表的连接就能查询出父子关系。

2. 查询每个人的父亲和母亲

以同样的方式查询父亲和母亲的信息，代码如下。

```
Select me.name  姓名,
          father.name  父亲,
          mother.name  母亲
     from family as me
          join family as father    on me.id_father=father.id
          join family as mother on me.id_mother=mother.id;
```

查询结果如图 5-41 所示，这时只包括父母双全的数据。

如果要包括父母已逝的所有人的信息，则代码如下，查询结果如图 5-42 所示。

```
Select me.name  姓名, me.sex  性别,
          father.name  父亲,
          mother.name  母亲
     from family as me
          left join family as father       on me.id_father=father.id
          left join family as mother on me.id_mother=mother.id;
```

5.4 聚合查询

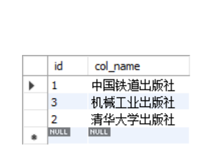
聚合查询是利用 MySQL 的聚合函数对数据进行统计和分组统计，常用的聚合函数如表 5-3 所示。

表 5-3 常用聚合函数

函数名	功能
avg(expression)	返回表达式中各值的平均值，只用于数字表达式，忽略 expressio 为 null 的值
sum(expression)	返回表达式中所有值的和，只用于数字表达式
min(expression)	返回表达式的最小值
max(expression)	返回表达式的最大值
count(expression)	返回结果的行数，忽略 expression 为 null 值的行
count(*)	返回结果的行数，包括 null 值



本节以图书销售数据为例，加上了销售数量和销售省份两列，进行聚合查询的讲解。出版社表（publisher）和图书表（book）的演示数据如图 5-43 所示，创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”。



id	col_name
1	中国铁道出版社
3	机械工业出版社
2	清华大学出版社
NULL	NULL

id	col_title	col_author	col_isbn	col_price	col_quantity	col_province	id_publisher
1	Microsoft SQL Server 2000宝典	[美]鲍尔	9787113057091	85.00	1	江苏省	1
2	数据库原理（第5版）	[美]大卫	9787302263432	49.80	1	上海市	2
3	SQL Server 2012数据库应用	李萍等编著	9787111505082	39.00	2	江苏省	3
4	MySQL数据库应用从入门到精通	崔洋等	9787113151317	59.80	1	上海市	1
5	MySQL DBA工作笔记:数据库管理、架构优化…	杨建荣编著	9787113260347	99.00	2	江苏省	1
6	数据库系统设计、实现与管理	Peter Rob著	9787302290124	69.00	3	上海市	2
7	数据库云平台理论与实践	马献章著	9787302421504	79.00	1	江苏省	2
8	MySQL8 数据库原理与实战	麻进玲等	9787111723639	45.00	2	上海市	3
9	MariaDB必知必会	Ben Forta著	9787111464280	59.00	2	上海市	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 5-43 出版社表（publisher）和图书表（book）的演示数据

5.4.1 无分组的统计

例如统计所有图书的数量、平均价格、最高价格和最低价格。

```
Use abc;                          -- 打开数据库 abc

Select count(*)  图书数量,
          round(avg(col_price), 2)  平均价格,
          min(col_price)  最低价格,
          max(col_price)  最高价格
     from book;
```

其中 round()是四舍五入函数，参数 2 表示保留 2 位小数。查询结果如图 5-44 所示。

5.4.2 分组统计 Group by

1. 一个分组列

可以通过 group by 子句进行分组统计，例如统计每家出版社图书的图书数量、平均价格、最高价格和最低价格。

```
Select id_publisher  出版社主键,
          count(*)  图书数量,
          round(avg(col_price), 2)  平均价格,
          min(col_price)  最低价格,
          max(col_price)  最高价格
     from book
     group by id_publisher;               -- 同一家出版社（id_publisher 相同）属于同一组
```

其中 group by 子句指定分组的依据。查询结果如图 5-45 所示，每一行的数据是每家出版社的统计数据。

图书数量	平均价格	最低价格	最高价格
9	64.96	39.00	99.00

图 5-44 简单的统计

出版社主键	图书数量	平均价格	最低价格	最高价格
1	3	81.27	59.80	99.00
2	3	65.93	49.80	79.00
3	3	47.67	39.00	59.00

图 5-45 分组统计

可以对表达式的结果进行统计，例如统计销售金额，代码如下，结果如图 5-46 所示。

```
Select id_publisher  出版社主键,
          sum(col_quantity)  销售数量,
          sum(col_quantity*col_price)  销售金额
     from book
     group by id_publisher;
```

出版社主键	销售数量	销售金额
1	4	342.80
2	5	335.80
3	6	286.00

图 5-46 以出版社为分组统计销售情况

出版社主键	销售省份	销售数量	销售金额
1	上海市	1	59.80
1	江苏省	3	283.00
2	上海市	4	256.80
2	江苏省	1	79.00
3	上海市	4	208.00
3	江苏省	2	78.00

图 5-47 以出版社和销售地区为分组统计销售情况

2. 两个分组列

可以根据多个列进行分组，例如根据出版社和销售省份进行分组，代码如下，结果如图 5-47 所示。

```
Select id_publisher  出版社主键,
          col_province  销售省份,
          sum(col_quantity)  销售数量,
          sum(col_quantity*col_price)  销售金额
     from book
     group by id_publisher, col_province
     order by id_publisher;
```

先根据出版社进行分组，然后在每家出版社内，再根据销售省份进行分组，从而得到更加详尽的统计结果。

5.4.3 筛选统计结果 Having

还可以对统计的结果进行筛选，去除不符合条件的统计结果，例如下述语句去除平均价格大于等于 80 元的行。

```
Select id_publisher  出版社主键,
          count(*)  图书数量,
          round(avg(col_price), 2)  平均价格,
```

```
        min(col_price) 最低价格,
        max(col_price) 最高价格
    from book
    group by id_publisher
        having 平均价格 < 80;
```

查询结果如图 5-48 所示。注意以下两点。

- Having 子句不应该单独出现，它必须紧接着 group by 子句，是对分组统计的结果作进一步的处理。如果 having 子句单独出现，应该将其合并到 where 子句中。
- Having 子句与 where 子句不同。where 子句是对统计前的数据进行筛选，不符合条件的行将不会参与统计，而 having 子句是对统计的结果进行筛选，不符合条件的结果行不会出现在结果集中。

出版社主键	图书数量	平均价格	最低价格	最高价格
2	3	65.93	49.80	79.00
3	3	47.67	39.00	59.00

图 5-48 Having 子句

出版社主键	图书数量	平均价格	最低价格	最高价格
2	3	65.93	49.80	79.00
3	3	47.67	39.00	59.00
1	1	59.80	59.80	59.80

图 5-49 Where 子句

例如下述语句只对价格小于 80 元的图书进行统计，运行结果如图 5-49 所示（特别注意参与统计的图书数量）。

```
Select id_publisher 出版社主键,
        count(*) 图书数量,
        round(avg(col_price), 2) 平均价格,
        min(col_price) 最低价格,
        max(col_price) 最高价格
    from book
    where col_price < 80
    group by id_publisher;
```

5.4.4 聚合查询与内连接

在前述分组统计中显示的分组是出版社主键，而不是出版社名称，参见图 5-47，当需要显示出版社名称时，需要与出版社表建立连接，在这个基础上进行分组统计。

```
Select col_name 出版社,
        col_province 销售省份,
        sum(col_quantity) 销售数量,
        sum(col_quantity*col_price) 销售金额
    from book join publisher
        on book.id_publisher = publisher.id
    group by id_publisher, col_province
    order by id_publisher;
```

运行结果如图 5-50 所示，与图 5-47 对比一下，图 5-50 提供的信息就直观多了。

出版社	销售省份	销售数量	销售金额
中国铁道出版社	上海市	1	59.80
中国铁道出版社	江苏省	3	283.00
清华大学出版社	上海市	4	256.80
清华大学出版社	江苏省	1	79.00
机械工业出版社	上海市	4	208.00
机械工业出版社	江苏省	2	78.00

图 5-50 分组统计与内连接

5.5 查询语句中的子句

前面几节讲解了 Select 语句各种子句的用法，这些子句必须以一定的次序出现，如下所示。

```
Select [all | distinct] {列名列表|*|表达式}
    from ...                    -- from 子句，指定表
        join ... on ...         -- join 子句，用于表的连接，子表.外键 = 主表.主键
```

```
        join … on …          -- join 子句，用于表的连接，子表.外键 = 主表.主键
    where …                  -- where 子句，指定查询条件
    group by …               -- group by 子句，指定统计时的分组
        having …             -- having 子句，用于对统计结果进行筛选，它从属于 group by 子句，
    order by …               -- order by 子句，指定排序列
    limit …;                 -- limit 子句，限制结果集的行数
```

所有子句必严格按上述次序出现。其中 join 子句用于表的连接，它从属于 from 子句，join 子句必须紧接于 from 子句之后，join 子句不能单独存在。having 子句从属于 group by 子句，having 子句必须紧接于 group by 子句之后，having 子句不应该单独存在。

可以省略不需要的子句，但是，除了 join 子句，其他子句都不能多次出现。



编写查询语句需要较高的技巧，同一个功能也可以采用多种方式实现，查询语句是本书的重点，在“6.1 子查询”和“9.5 查询技巧”还有更多的讲解。

【案例讲解】书店管理系统的查询



附录 D 项目 2c

打开附录 D 的“项目 2c 书店管理项目”，对照项目的运行过程，重点学习有关数据查询的内容。在演示版的体验过程中，请关注下述三方面的内容。

- 首先了解项目的运行，每个菜单的功能是什么，显示哪张表或哪些表的数据，是否有新增、编辑和删除功能。
- 当运行某个菜单的功能时，想一想后台执行了什么 SQL 语句，尝试自行编写这个功能的语句，然后在“运行日志”中找到后台实际执行的语句，加以验证。
- 如果想要看看这些功能是如何设计出来的，可以切换到开发版，进入开发模式，这时可以看到 SQL 语句是如何编写的。



在继续本【案例讲解】的学习前，请读者先学习一下单元 9 的“9.1 实战演练平台的安装和使用”和“9.2 图书信息项目的开发”。

任务 1 使用单表查询

这里以“图书管理”功能为例，图书管理的界面参见单元 2 的图 2-20，它的上半部分显示出版社表的列表，下半部分显示选中的出版社所出版的图书的列表。

查询出版社表的查询语句如下。

```
Select * from bs_publisher;
```

查询图书的查询语句如下。

```
Select * from bs_book where id_bs_publisher = {$rowId};
```

其中{$rowId}将会被选中的出版社的 ID 值替换，用户每选择一次出版社，上述查询语句会被执行一次，从而显示当前选中出版社的图书列表。在“运行日志”中显示的语句可能如下所示。

```
Select * from bs_book where id_bs_publisher = 1;
Select * from bs_book where id_bs_publisher = 2;
Select * from bs_book where id_bs_publisher = 1;
Select * from bs_book where id_bs_publisher = 1;
```

上述 4 条查询语句对应了用户的 4 次选择（单击出版社），其中 ID 为 1 的出版社共选择了 3 次。

任务 2 使用连接查询

这里以“订单录入”为例，订单录入的主表设计的主表数据源语句如下。

```
Select bs_order.*, col_name, col_tel, bs_customer.col_addre mr from bs_order
    join bs_customer on bs_order.id_bs_customer = bs_customer.id
```

```
where col_status = '订单' or col_status = '已收款';
```

这是因为订单中还需要显示客户的许多信息。从表设计的从表数据源语句如下。

```
Select bs_order_detail.*, col_author, col_title, col_name, col_isbn, col_year from bs_order_detail
    join bs_book on bs_order_detail.id_bs_book = bs_book.id
    join bs_publisher on bs_book.id_bs_publisher = bs_publisher.id
where id_bs_order = {$rowId};
```

这是因为订单明细中还需要显示图书的信息，和出版社的信息，所以把 3 张表连接起来。

任务 3 使用聚合查询

这里以“按出版社统计功能”为例，按出版社统计的界面见图 5-51，它显示每家出版社的销售数量和销售金额。



图 5-51 按出版社统计销售情况

按出版社统计需要进行分组统计，并且还要通过连接查询，列出出版社的名称，代码如下。

```
Select bs_publisher.col_name name, sum(col_quantity) quantity, sum(col_ammount) ammount from bs_order_detail
        join bs_book on bs_order_detail.id_bs_book = bs_book.id
        join bs_publisher on bs_book.id_bs_publisher = bs_publisher.id
        join bs_order on bs_order_detail.id_bs_order = bs_order.id
    where bs_order.col_status = '已发货'
    group by bs_publisher.id;
```

由于统计的是实际销售，所以只能统计“已发货”的订单，其筛选条件应该在 where 子句中，而不是 having 子句中。

项目的其余三项统计“按客户统计”、“按发货日期统计”和“按月份销售统计”的代码有错误，请读者切换到开发版，修改代码，完成这三项统计功能。

【实战演练】图书借阅系统的查询



参考附录 D 的“项目 2d 图书借阅项目”，在读者自行开发的图书借阅项目上，根据项目的需求，编写“基础资料”的 3 个模块和“借还书”管理的 2 个模块中的查询语句，并尝试进行该项目的开发。



“项目 2d 图书借阅项目”演示版的代码不对读者开放，读者需要参考演示版的功能，在自行设计的数据结构（在单元 2 和单元 3 的实战演练中完成）的基础上进行项目的开发。

如果有些代码写不出来，可以参考项目 2d 的“运行日志”，看看在项目 2d 中，相同功能的代码是怎么写的。

如果开发中有疑问，可以参考单元 9 的“9.2 图书信息项目的开发”，开发“图书借阅系统”只需要编写 Select 语句，无需其他任何语句。

【单元重点】

单元小结参见本单元的【思维导图】，单元重点如下。

- 单表查询中的所有内容都是重点，是所有查询的基础，必须熟练掌握。
- 联合查询应该熟练掌握。

- 连接查询的内连接是重中之重，必须熟练掌握。
- 聚合查询在实际项目中经常用到，需要熟练掌握。

数据查询是 SQL 的灵魂，数据定义、数据操纵都是为数据查询服务的，本单元是数据查询的基础，是本书重点中的重点，在“6.1 子查询”和“9.5 查询技巧”还有更多的讲解。

【课后思考】

一、选择题

1. 列的别名在什么情况下必须用引号括起来【　　　　】。
   A. 别名含有汉字　B. 别名含有空格　C. 别名含有特殊字符　D. 以上都是
2. 关键字 distinct 用于去除重复行，什么样的行是重复行（　　）。
   A. 主键值相同的行　B. 指定列的值相同的行
   C. 所有列的值都相同的行　D. 以上都是
3. 计算列中可以使用（　　）。
   A. 常量　B. 表达式　C. 函数　D. 以上都是
4. 范围查询使用的关键字是【　　　　】。
   A. between…and…　B. in　C. like　D. where
5. 集合查询使用的关键字是【　　　　】。
   A. between…and…　B. in　C. like　D. where
6. 模糊查询使用的关键字是【　　　　】。
   A. between…and…　B. in　C. like　D. where
7. 模糊查询使用通配符进行匹配，其中匹配 0 到多个字符的符号是（　　）。
   A. *　B. %　C. ?　D. _
8. 模糊查询使用通配符进行匹配，其中匹配正好一个字符的符号是（　　）。
   A. *　B. %　C. ?　D. _
9. 空值判断的运算符是（　　）。
   A. = null　B. null　C. is null　D. not null
10. Order by 子句中指定降序排序的关键字是（　　）。
    A. asc　B. desc　C. order　D. union
11. 联合查询使用的关键字是（　　）。
    A. asc　B. desc　C. order　D. union
12. 在省略了 from 子句的情况下，想要得到两行或以上查询结果，需要使用的关键字是（　　）。
    A. desc　B. from　C. join　D. union
13. 连接查询使用的关键字是（　　）。
    A. desc　B. from　C. join　D. union
14. 与 join 同时使用的子句是（　　）。
    A. desc　B. from　C. join　D. union
15. 在内连接查询的两张表中，这两张表的联系应该是【　　　　】。
    A. 一对一联系　B. 一对多联系　C. 多对多联系　D. 可以没有联系
16. 求平均值的聚合函数是（　　）。

A. avg　　B. count　　C. min　　D. sum

17. 无分组统计的查询结果的行数（　　）。

A. 正好 1 行　　B. 0 或 1 行　　C. 0 到多行　　D. 1 到多行

18. 有分组统计（group by）的查询结果的行数（　　）。

A. 正好 1 行　　B. 0 或 1 行　　C. 0 到多行　　D. 1 到多行

19. 通常与 having 同时使用的子句是（　　）。

A. from　　B. group by　　C. order by　　D. limit

20. 一条查询语句可以包含哪些查询（　　）。

A. 联合查询　　B. 连接查询　　C. 聚合查询　　D. 以上全部

二、填空题

21. 如果 A 表有 3 行，B 表有 6 行，A 表和 B 表可能有相同的数据，则 Select … from A union select … from B 的结果最少是_____行，最多是_____行。

22. 如果 A 表有 3 行，B 表有 6 行，A 表和 B 表可能有相同的数据，则 Select … from A union ***all*** select … from B 的结果是_____行。

23. 如果 A 表有 3 行，B 表有 6 行，则 A join B 的结果最少是_____行，最多是_____行。

三、思考题

1. 联合查询与连接查询有什么区别？

2. Where 子句与 having 子句有什么区别？

【课外拓展】

3、 问一问 AI，比较内连接与等值连接的区别，并谈谈用哪一种更合适一些。

4、 模糊查询是用通配符进行匹配的查询，MySQL 还提供了一种更为强大的查询，通过正则表达式（regexp）进行查询，请查找相关资料，对正则表达式进行初步的了解。

5、 从网上查找名为“阿里巴巴 Java 开发手册-2022.pdf”的文件，阅读其中关于 SQL 语句的部分。

【提高篇】

本书的单元 6~单元 9 是提高篇，多数二级标题可以独立成篇，读者可以选择性地学习。

在学习提高篇之前，建议先学习“单元 9 项目实战”的“9.2 图书信息项目的开发”，以便通过实战项目加深对基础篇内容的理解，为学习提高篇打下更扎实的基础。

单元6 子查询、视图和索引

【学习目标】

知识目标

- ◆ 理解简单子查询和相关子查询。
- ◆ 理解视图的特点和作用。
- ◆ 理解索引的作用、分类和设计原则。

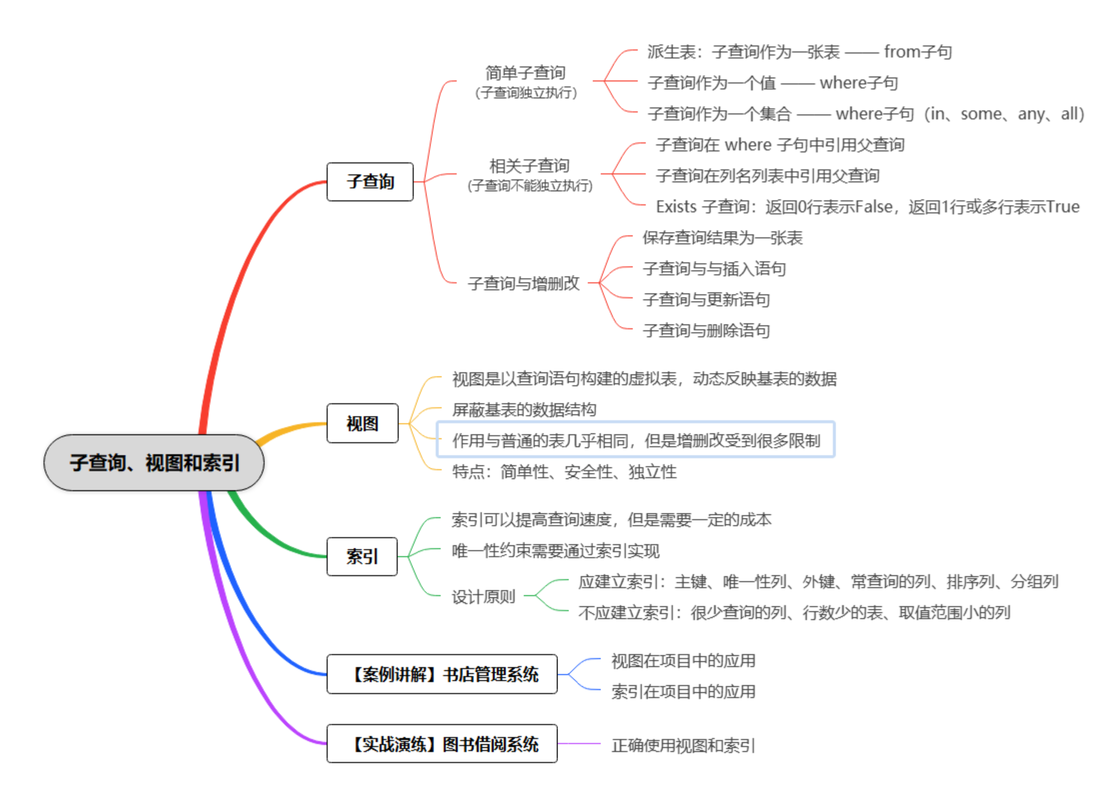
能力目标

- ◆ 初步学会使用简单子查询和相关子查询。
- ◆ 初步学会增删改与子查询的联合使用。
- ◆ 学会创建和管理视图。
- ◆ 学会创建和管理索引。

素质目标

- ◆ 提高逻辑思维能力、嵌套、递归、抽象。
- ◆ 团队精神、效率意识。

【思维导图】



【情景导入】

小明学完了数据查询，体验到数据库的丰富功能，也理解了为什么在数据库设计时要拆分实体，拆分后的表通过连接查询可以再连接起来，满足用户的各种需求，而 where、order by、limit、group by 子句又使查询结果灵活多变。小明还想继续学习数据查询的更多技能，让我们与小明一起继续吧。

【知识储备】

单元 1～单元 5 分别讲解了数据库管理系统（DBMS）、关系数据库的理论知识、数据定义（DDL）、数据操纵（DML）和数据查询（DQL），这是本书的核心和精华所在。在接下来的单元 6～单元 9，将重点讲解子查询、视图、索引、数据库编程、数据库管理和项目实战等专题，多数二级标题的内容都可以独立成篇，读者可以选择学习。

数据查询是关系型数据库的核心功能，本单元通过子查询，对数据查询再进行一些深入的讲解，并简单讲解与数据查询有关的视图和索引方面的知识。

6.1 子查询

子查询是指被包含在一条语句中的查询语句，这条被包含的查询语句就称为子查询，包含子查询的语句可以是查询语句，也可以是插入语句、更新语句或删除语句。

最常见的子查询是被包含在查询语句中的，就是在 Select 语句（父查询）中还有一个 Select 查询（子查询），子查询可以作为一张表、一个值、一个集合（多行一列）而出现在父查询的 from、where 等子句中，甚至还能出现在列名列表中，如图 6-1 所示，子查询必须用圆括号括起来。

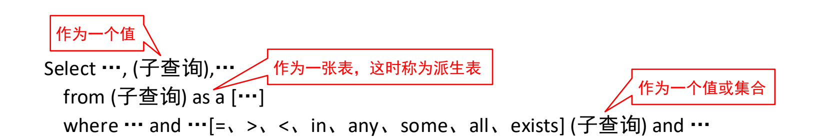


图 6-1 子查询在父查询中的位置



查询的结果永远是一张表，可能是 1 行 1 列的表（相当于一个值），也可能是多行 1 列的表（相当于一个集合），一般情况下是多行多列的表。

本节使用“5.4 聚合查询”的出版社表和图书表数据进行讲解。

6.1.1 简单子查询



简单子查询是一种简单的子查询，它的简单性表现在子查询可以独立执行，父查询仅仅是使用了子查询的结果。

1. 派生表 —— 子查询作为一张表出现在 From 子句中

首先看看下述查询的结果，它计算每种图书的销售金额（即价格*数量），结果如图 6-2 所示。

```
Select *, col_price*col_quantity amount
    from book;
```

id	col_title	col_author	col_isbn	col_price	col_quantity	col_province	id_publisher	amount
1	Microsoft SQL Server 2000宝典	[美]鲍尔	9787113057091	85.00	1	江苏省	1	85.00
2	数据库原理（第5版）	[美]大卫	9787302263432	49.80	1	上海市	2	49.80
3	SQL Server 2012数据库应用	李萍等编著	9787111505082	39.00	2	江苏省	3	78.00
4	MySQL数据库应用从入门到精通	崔洋等	9787113151317	59.80	1	上海市	1	59.80
5	MySQL DBA工作笔记:数据库管理、架构优化…	杨建荣编著	9787113260347	99.00	2	江苏省	1	198.00
6	数据库系统设计、实现与管理	Peter Rob著	9787302290124	69.00	3	上海市	2	207.00
7	数据库云平台理论与实践	马献章著	9787302421504	79.00	1	江苏省	2	79.00
8	MySQL8 数据库原理与实战	麻进玲等	9787111723639	45.00	2	上海市	3	90.00
9	MariaDB必知必会	Ben Forta著	9787111464280	59.00	2	上海市	3	118.00

图 6-2 查询每种图书的销售金额

1) 问题的提出

现在需要编写一条查询销售金额大于 100 元的图书，代码如下所示。

```
Select *, col_price*col_quantity amount
    from book
    where amount > 100;
```

但是这条语句执行时出错，出错信息如下。

```
Error Code: 1054. Unknown column 'amount' in 'where clause'
```

出错信息的意思是 where 子句中的 amount 列找不到，这是因为别名是不能用于 where 子句中的。

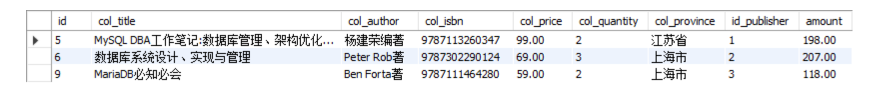
2) 采用子查询解决问题

为了解决这个问题，可以将这条语句写成子查询，在父查询中就可以找到 amount 列，代码如下所示

（子查询必须用括号括起来）。

```
Select *
    from
    (Select *, col_price*col_quantity amount
        from book
    ) a
    where amount > 100;
```

上述语句中，括号内查询语句的运行结果是一张表，即如图 6-2 所示的表，它有销售金额列（amount），再对这张表进行查询，在父查询的 where 子句就能找到 amount 列了，查询结果如图 6-3 所示。



id	col_title	col_author	col_isbn	col_price	col_quantity	col_province	id_publisher	amount
5	MySQL DBA工作笔记:数据库管理、架构优化…	杨建荣编著	9787113260347	99.00	2	江苏省	1	198.00
6	数据库系统设计、实现与管理	Peter Rob著	9787302290124	69.00	3	上海市	2	207.00
9	MariaDB必知必会	Ben Forta著	9787111464280	59.00	2	上海市	3	118.00

图 6-3 查询销售金额大于 100 的图书



这种子查询称为派生表，它必须要有一个别名，否则会出错。这个例子中派生表的别名是 a，尽管这个别名常常没有任何作用。

2. 子查询作为一个值出现在 Where 子句中

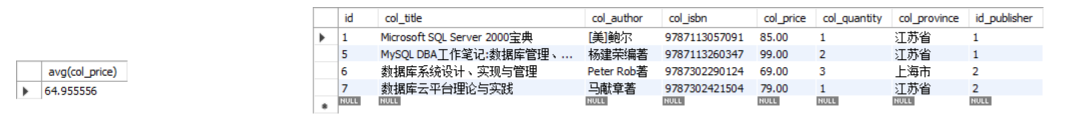
这个例子的需求是查询价格大于所有图书平均价格的图书。

1) 不使用子查询的解决方案

为实现这个目标，先查询所有图书的平均价格。

```
Select avg(col_price)
    from book;
```

查询的结果是平均价格为 64.955556 元，如图 6-4 所示。



avg(col_price)
64.955556

图 6-4 查询平均价格

id	col_title	col_author	col_isbn	col_price	col_quantity	col_province	id_publisher
1	Microsoft SQL Server 2000宝典	[美]鲍尔	9787113057091	85.00	1	江苏省	1
5	MySQL DBA工作笔记:数据库管理、…	杨建荣编著	9787113260347	99.00	2	江苏省	1
6	数据库系统设计、实现与管理	Peter Rob著	9787302290124	69.00	3	上海市	2
7	数据库云平台理论与实践	马献章著	9787302421504	79.00	1	江苏省	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 6-5 查询价格大于平均价格的图书

然后再写一条如下的查询语句，所得结果就是价格大于平均价格的图书，结果是一共有 4 种图书，属于两个出版社，另一个出版社的图书不在其中，因该社的图书价格都比较低，如图 6-5 所示。

```
Select *
    from book
    where col_price > 64.955556;
```

上面用了两条查询语句，并且是手动地将平均价格从第一条语句的执行结果中复制到第二条语句的查询条件中。

2) 子查询的解决方案

一个完美的解决办法是将这两条语句合并在一起，形成一条嵌套的查询语句。就是说，将第一条语句（整条 Select 语句）复制到第二条语句的查询条件中，而不是复制第一条语句的执行结果（平均价格），避免了手动复制时可能的错误。代码如下（子查询要用一对括号括起来），查询结果与图 6-5 完全相同。

```
Select *
    from book
    where col_price >
        (Select avg(col_price)
            from book
        );
```

父查询 where 条件的运算符是比较运算符（>），比较运算符的两边都必须是标量，因此子查询必须

只能返回 1 行 1 列，相当于是返回一个值。另外，where 子句中的子查询应该放在比较运算符的右侧。

3. 子查询作为一个集合出现在 Where 子句中

子查询作为一个集合时，还分为两种情况，下面分别讲解。

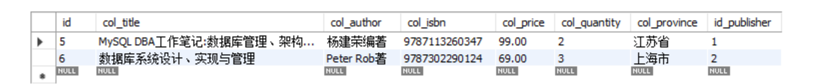
1) In 子查询

例如要查询出版的图书书名中含有“管理”字样的出版社信息。可以先查询书名中含有“管理”字样的图书，代码如下，查询结果如图 6-6 所示。

```
Select * from book
        where col_title like '%管理%';
```

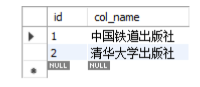
结果中的出版社 ID 是 1 和 2，再用下述语句查询这两家出版社的信息，查询结果如图 6-7 所示。

```
Select * from publisher
        where id in (1, 2);
```



id	col_title	col_author	col_isbn	col_price	col_quantity	col_province	id_publisher
5	MySQL DBA工作笔记:数据库管理、架构...	杨建荣编著	9787113260347	99.00	2	江苏省	1
6	数据库系统设计、实现与管理	Peter Rob著	9787302290124	69.00	3	上海市	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 6-6 查询



id	col_name
1	中国铁道出版社
2	清华大学出版社
NULL	NULL

图 6-7 查询

将上述第一条语句复制到第二条语句中，就成为了 in 子查询，代码如下，查询结果与图 6-7 相同。

```
Select * from publisher
        where id in (select id_publisher from book
                where col_title like '%管理%'
        );
```

2) Any、some、all 子查询

Any、some、all 子查询还需要与比较运算符（<、<=、>、>= 等）协同工作，因此也称为比较子查询。

这里以 all 子查询为例讲解。先看看下述语句的查询结果，它计算每家出版社的图书的平均价格，查询结果如图 6-8 所示。

```
Select id_publisher, avg(col_price)
        from book
        group by id_publisher;
```

id_publisher	avg(col_price)
1	81.266667
2	65.933333
3	47.666667

图 6-8 每家出版社的图书的平均价格

id	col_title	col_author	col_isbn	col_price	col_quantity	col_province	id_publisher
3	SQL Server 2012数据库应用	李萍等编著	9787111505082	39.00	2	江苏省	3
8	MySQL8 数据库原理与实战	麻进玲等	9787111723639	45.00	2	上海市	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 6-9 查询价格小于所有出版社平均价格的图书

现在要查询小于所有出版社的图书平均价格的图书，就是价格小于 81.266667 和 65.933333 以及 47.666667 的图书，即小于所有（all）这三个价格的图书，代码如下，查询结果如图 6-9 所示。

```
Select *
        from book
        where col_price < all
                (Select avg(col_price)
                        from book
                        group by id_publisher
                );
```

父查询 where 条件的运算符含有关键字 all，这个关键字允许子查询返回多行，但是只允许 1 列，所以要把图 6-8 所示的结果中移除 id_publisher 列，这时相当于是返回一个集合。

4. 简单子查询的特点

简单子查询的特点是子查询本身可以独立执行。

简单子查询的执行过程是先执行子查询，然后再执行父查询，一共执行两次查询，子查询的结果被父

查询使用。这与下面讲解的相关子查询有很大的不同。

6.1.2 相关子查询

相关子查询比较复杂一些，子查询不能独立执行，子查询需要引用父查询的表。

Jitor 实训 | 附录 C 实训 6-2

1. 子查询在 where 子句中引用父查询

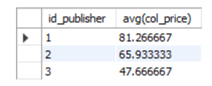
在上一小节的简单子查询中，查询价格大于平均价格的图书，结果一共有 4 种图书，属于两个出版社，有一个出版社的图书不在其中（参见图 6-5），因为该社的图书价格都比较低。

现在把查询的要求作一些修改，查询价格大于本社图书平均价格的图书，这样每家出版社都会有图书入选。这两个查询的区别在于前者是查询价格大于“所有图书”平均价格的图书，后者是查询价格大于“本社图书”平均价格的图书。

1) 不使用子查询的解决方案

为此，先查询每家出版社的图书平均价格，结果如图 6-10 所示。

```
Select id_publisher, avg(col_price)
        from book
        group by id_publisher;
```



id_publisher	avg(col_price)
1	81.266667
2	65.933333
3	47.666667

图 6-10 每家出版社的图书平均价格

再分别查询每家出版社价格大于“本社图书平均价格”的图书，并将结果联合起来，语句如下所示。

```
Select *
        from book
        where col_price > 81.266667
                and id_publisher = 1
Union
Select *
        from book
        where col_price > 65.933333
                and id_publisher = 2
Union
Select *
        from book
        where col_price > 47.666667
                and id_publisher = 3;
```

查询结果如图 6-11 所示，这时每家出版社都有图书入选。

id	col_title	col_author	col_isbn	col_price	col_quantity	col_province	id_publisher
1	Microsoft SQL Server 2000宝典	[美]鲍尔	9787113057091	85.00	1	江苏省	1
5	MySQL DBA工作笔记:数据库管理、架构优化...	杨建荣编著	9787113260347	99.00	2	江苏省	1
6	数据库系统设计、实现与管理	Peter Rob著	9787302290124	69.00	3	上海市	2
7	数据库云平台理论与实践	马献章著	9787302421504	79.00	1	江苏省	2
9	MariaDB必知必会	Ben Forta著	9787111464280	59.00	2	上海市	3

图 6-11 查询价格大于本社图书平均价格的图书

2) 子查询的解决方案

可以将上述两条语句合并为一条语句，避免手工复制中间数据，合并后的代码如下所示，查询结果与图 6-11 所示完全相同。

```
Select *
        from book parent
        where col_price >
                (Select avg(col_price)
                        from book
```

```
        where id_publisher = parent.id_publisher
    );
```

注意在父查询中为 book 表定义了一个别名 parent（含义是父表），在子查询的 where 子句引用了父表的别名 parent，通过别名将子查询和父查询关联起来，成为相关子查询。

2. 子查询在列名列表中引用父查询

子查询还可以出现在列名列表中。例如下述代码，出版社列是一个返回 1 行 1 列的子查询，子查询引用了父查询 book 表的外键。

```
Select *,
    (Select col_name
        from publisher where id=book.id_publisher
    ) 出版社
  from book;
```

运行结果如图 6-12 所示，其效果与内连接相同。在这种情况下，内连接的效率会高一些，因此应该尽量用内连接。

id	col_title	col_author	col_isbn	col_price	col_quantity	col_province	id_publisher	出版社
1	Microsoft SQL Server 2000宝典	[美]鲍尔	9787113057091	85.00	1	江苏省	1	中国铁道出版社
2	数据库原理（第5版）	[美]大卫	9787302263432	49.80	1	上海市	2	清华大学出版社
3	SQL Server 2012数据库应用	李萍等编著	9787111505082	39.00	2	江苏省	3	机械工业出版社
4	MySQL数据库应用从入门到精通	崔洋等	9787113151317	59.80	1	上海市	1	中国铁道出版社
5	MySQL DBA工作笔记:数据库管...	杨建荣编著	9787113260347	99.00	2	江苏省	1	中国铁道出版社
6	数据库系统设计、实现与管理	Peter Rob著	9787302290124	69.00	3	上海市	2	清华大学出版社
7	数据库云平台理论与实践	马献章著	9787302421504	79.00	1	江苏省	2	清华大学出版社
8	MySQL8 数据库原理与实战	麻进玲等	9787111723639	45.00	2	上海市	3	机械工业出版社
9	MariaDB必知必会	Ben Forta著	9787111464280	59.00	2	上海市	3	机械工业出版社

图 6-12 子查询在列名列表中的查询结果

3. Exists 子查询

使用关键字 exists（以及 not exists）的子查询通常属于相关子查询。

下述代码查询销售数量大于 1 的图书（最简单高效的办法是父查询直接用 where col_quantity > 1，这个例子仅仅是为了演示 exists 子查询）。

```
Select *
  from book parent
  where exists
  (Select *
      from book
      where id = parent.id and col_quantity > 1
  );
```

Exists 子查询有如下特点：

- 父查询只关心子查询返回的行数，返回 0 行表示 False，返回 1 行或多行表示 True。因此子查询中的列名列表通常使用星号（*）。
- 父查询根据子查询返回 False 还是 True 来决定结果集中是否包含该行。
- Exists 子查询通常会引用父查询的表，因此也是一种相关子查询。

4. 相关子查询的特点

相关子查询的特点是子查询本身不能独立执行，子查询需要引用父查询的表。根据这个特点就很容易区分简单子查询和相关子查询。

相关子查询的执行过程是先执行父查询，然后根据父查询的查询结果，再多次执行子查询，因此执行查询的次数多于 2 次。

简单子查询和相关子查询的区别如表 6-1 所示。

表 6-1 简单子查询和相关子查询的区别

比较项	简单子查询	相关子查询
子查询是否引用父查询的表	否	是，通常父查询的表需要指定别名
子查询能否独立执行	能独立执行	不能独立执行
先执行哪条查询	子查询	父查询
子查询的执行次数	1 次	N 次，N 是父查询结果的行数

6.1.3 子查询与增删改



前述两小节的子查询是作为 Select 语句的子查询，子查询还能够与 Insert 语句、Update 语句、Delete 语句甚至是 Create table 语句配合使用，如图 6-13 所示，大大增强增删改语句的功能。

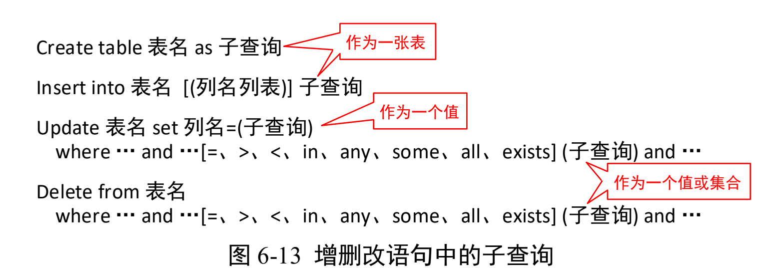


图 6-13 增删改语句中的子查询

1. 保存查询结果为一张表

查询的结果是一张二维表，因此可以将查询结果保存为一张表。语法格式如下。

```
Create table 表名 as 子查询;
```

因为查询结果只包含列名和数据类型，不包含非空约束和默认约束之外的数据约束（查询结果可能有主键列，但没有主键约束），所以新表没有主键约束等数据约束，也没有索引。

例如下述语句保存 book 表的数据为 book1 表。

```
Use abc;                    -- 打开数据库 abc，本小节使用“5.4 聚合查询”的图书表和出版社表
Create table book1 as       -- 保存查询结果为 book1 表，book1 表的 id 列没有主键约束
    Select * from book;
```

子查询可以是任何查询，包括多表查询。例如下述代码保存 book 表的部分数据为一张新的表。

```
Create table book2 as       -- 查询结果只有部分列和部分行
    select col_title title, col_author author, col_price price from book
        where col_title like '%MySQL%';
```

如果只想保存列名和数据类型，而不想同时保存数据，则可以改为如下代码。

```
Create table book3 as               -- 保存查询结果为 book3 表，不含数据
    Select * from book where 1=2;   -- 查询结果中不含数据，条件 1=2 永远不成立
```

2. 子查询与插入语句

1) 插入子查询的数据

可以向一张表插入子查询的结果。语法格式如下。

```
Insert into 表名 [(列名列表)] 子查询;
```

例如下述代码向 publisher 表插入来自于 book 表的数据（数据含义是错的）。

```
Insert into publisher (col_name)    -- 向 publisher 插入子查询的结果
    Select col_title from book      -- 含义是错的：将书名作为出版社的名称，插入到出版社表中
        where id<3;
```

与普通插入语句一样，这时仍然要求插入的列与查询的列在数量、类型和含义上完全相同。

2) 复制表

前述“保存查询结果为一张表”的第一个例子似乎是复制了一张表，但是新表只包含查询结果中的信息，而没有完整的数据约束和索引等。

想要完全复制一张表，要采用如下两条语句。

```
Create table book_1 like book;          -- 先复制表，含所有数据约束和索引，但不含数据
Insert into book_1                      -- 再复制数据，数据来自于子查询的结果
        Select * from book;
```

上述两条语句完整地复制了一张表，Create 语句复制表结构（参见“3.3.6 复制表”，含所有数据约束和索引），Insert 语句复制数据，结果是两张表拥有完全相同的结构和数据。

3. 子查询与更新语句

子查询可以出现在更新语句中的 where 子句或赋值的表达式中。

Where 子句例子与简单子查询和相关子查询中 where 子句的例子基本相同。这里用一个例子讲解用于赋值表达式中，先查询每家出版社的销售金额，代码如下所示。

```
Select id_publisher, sum(col_price*col_quantity)        -- 销售金额 = 价格*销售数量
        from book
        group by id_publisher;
```

查询结果如图 6-14 所示。现在为出版社表添加销售金额列 col_amount，然后将如图 6-14 所示的数据更新到出版社表中，代码如下所示。

```
Alter table publisher
        add column col_amount decimal(9,2) not null default 0;        -- 添加金额列 col_amount

Update publisher
        set col_amount =
                (Select sum(col_price*col_quantity)
                        from book
                        where id_publisher = publisher.id
                );
```

上述代码中，更新语句所赋的值是从 book 表中查询而来的，在子查询中引用了更新语句的表，因此它的运行机制类似于相关子查询。更新后，出版社表中的销售金额数据如图 6-15 所示。



id_publisher	sum(col_price*col_quantity)
1	342.80
2	335.80
3	286.00

图 6-14 查询每家出版社的销售金额

id	col_name	col_amount
1	中国铁道出版社	342.80
2	清华大学出版社	335.80
3	机械工业出版社	286.00
NULL	NULL	NULL

图 6-15 出版社表中的销售金额数据

4. 子查询与删除语句

子查询可以出现在删除语句的 where 子句中。

用一个例子加以说明，这个例子是从 book 表中删除销售金额小于 300 元的出版社的图书，从图 6-15 可以看到，是要删除 book 表中出版社 id 为 3 的图书，代码如下所示。

```
Delete from book
        where id_publisher in
                (Select id from publisher
                        where col_amount < 300
                );
```

上述代码中的子查询将会查询得到销售金额小于 300 的出版社 id，不论子查询的结果有多少行，外层的删除语句将删除图书表中 id_publisher 在这个 id 组成的集合中的行。

6.2 视图

Select 查询的结果是一张二维表，与普通的表在许多方面是相同的，因此可以为 select 语句进行命名，这个名字与查询结果相对应，这就是视图。

视图是虚拟的表，其作用与普通的表几乎是完全相同的，核心的区别是视图并不实际保存数据，数据来源是 select 语句 from 子句中涉及的表，这些表被称为视图的基表，基表中数据的改变将动态地反映到视图中。

6.2.1 视图的特点

视图与表在许多方面是相同的，但有如下区别。

- 表保存的是实际数据，视图保存的是预编译的 select 语句。
- 表的结构和数据是物理存在的，而视图只是一个虚表。
- 基表中的数据发生变化时，从视图中查询出的数据也随之改变。
- 修改基表结构时，如果涉及到视图的 select 语句，则视图也需要修改。
- 表可以进行增删改操作，而视图只能在有限条件下进行增删改操作。

因此，表是内容，视图是观察内容的窗口，从数据库系统的体系结构来看，表是模式，视图是外模式（参见单元 2【课外拓展】中的问题），表属于全局模式中的表，是实表，视图属于局部模式的表，是虚表。视图是一个简单但却是一个十分有用的工具，受到广泛的应用。视图的优点如下。

- 简单性：视图可以简化对数据的理解，也可以简化对数据的操作。可以将经常使用的复杂的连接查询定义为视图，在使用时不必每次指定连接操作等复杂的子句，简化查询语句的编写。
- 安全性：通过视图可以屏蔽某些数据，也可以只赋予特定的用户查看特定数据的权限，而对其他数据既无法查看，更无法修改，从而保障数据的安全。
- 独立性：视图可以屏蔽基表结构变化带来的影响。如果基表的结构发生变化，可以修改视图，而使视图的功能保持不变，从而保持应用程序不变。

6.2.2 创建视图



附录 C 实训 6-4

创建视图的语法格式如下。

```
Create view <视图名>
as
        <Select …>;
```



- 视图名：在数据库范围内唯一的标识符，通常以 v_起头。
- 视图中 Select 语句中的每列必须有唯一的列名，不允许出现二义性的列名（不同表的同名列），也不允许出现未定义的列名（无别名的计算列）。
- 视图中 Select 语句不能有 order by 子句。
- 基表的结构改变时，如果改变的部分涉及到视图的 Select 语句，则必须重建视图。

本节与上节一样，仍使用“5.4 聚合查询”的出版社表和图书表数据进行讲解。例如下述语句将图书表和出版社表连接起来，查询结果如图 6-16 所示。

```
Select book.id, col_title, col_author, col_isbn, col_price, col_name
        from book
                join publisher on book.id_publisher = publisher.id;
```



id	col_title	col_author	col_isbn	col_price	col_name
1	Microsoft SQL Server 2000宝典	[美]鲍尔	9787113057091	85.00	中国铁道出版社
2	数据库原理（第5版）	[美]大卫	9787302263432	49.80	清华大学出版社
3	SQL Server 2012数据库应用	李萍等编著	9787111505082	39.00	机械工业出版社
4	MySQL数据库应用从入门到精通	崔洋等	9787113151317	59.80	中国铁道出版社
5	MySQL DBA工作笔记:数据库管…	杨建荣编著	9787113260347	99.00	中国铁道出版社
6	数据库系统设计、实现与管理	Peter Rob著	9787302290124	69.00	清华大学出版社
7	数据库云平台理论与实践	马献章著	9787302421504	79.00	清华大学出版社
8	MySQL8 数据库原理与实战	麻进玲等	9787111723639	45.00	机械工业出版社
9	MariaDB必知必会	Ben Forta著	9787111464280	59.00	机械工业出版社

图 6-16 图书表（含出版社名称）

如图 6-16 所示的数据是一张表，将这张表作为视图进行命名，代码如下所示。

```
Create view v_book
as
Select book.id, col_title, col_author, col_isbn, col_price, col_name
	from book
		join publisher on book.id_publisher = publisher.id;
```

上述代码创建了一个名为 v_book 的视图。

6.2.3 使用视图

创建了 v_book 视图后，相当于在数据结构上增加了一张名为 v_book 的表，它的数据结构是在图书表的基础上，加上了出版社名称。它是一张虚拟表，数据来自于图书表和出版社表。



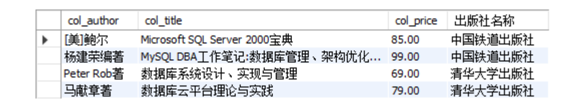
本单元案例讲解中有一个视图的应用。单元 9“9.3 视图的应用”中也有一个视图的应用，同一个视图还在“9.4 事务的应用”的事务中使用，因此具有十分典型的意义。

1. 查询中使用视图

视图可以用在 select 语句中使用表的任何地方，在这个方面视图的作用与表完全相同。例如下述代码使用前面创建的视图 v_book。

```
Select col_author, col_title, col_price, col_name 出版社名称
	from v_book
	where col_price > 60;
```

在上述代码中不需要进行连接操作，就能够查询到出版社的信息，这是因为视图 v_book 已经包含了表的连接。查询结果如图 6-17 所示。



col_author	col_title	col_price	出版社名称
[美]鲍尔	Microsoft SQL Server 2000宝典	85.00	中国铁道出版社
杨建荣编著	MySQL DBA工作笔记:数据库管理、架构优化…	99.00	中国铁道出版社
Peter Rob著	数据库系统设计、实现与管理	69.00	清华大学出版社
马献章著	数据库云平台理论与实践	79.00	清华大学出版社

图 6-17 查询视图的结果

2. 增删改中使用视图

视图可以在有限的条件下通过 insert、update、delete 语句对基表进行增删改操作。这些条件如下。

- 定义视图的 select 语句中不能含有计算列（常量、表达式、函数，如 sum()等）
- 定义视图的 select 语句中不能含有 union、distinct、group by、having 等关键字。
- 定义视图的 select 语句中不能含有子查询。
- 视图基表中的无默认值非空列必须包含在视图 select 语句的列名列表中。
- 如果定义视图的 select 语句中含有 join 子句，最多只能对其中一张基表进行增删改操作。



虽然可以通过视图对基表进行增删改操作，但是限制条件太多，一般并不常用。因此，视图主要是用在 select 语句中。

6.2.4 管理视图

视图是一种数据库对象，因此创建视图后，可以在 Workbench 的导航栏看到创建的视图（可能需要刷新一下），如图 6-18 所示。

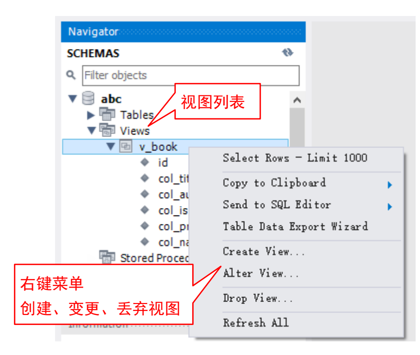


图 6-18 导航栏中的视图



图 6-19 查看表和视图的列表

1. 列出视图及视图结构

视图与表在许多方面是相同的，也可以列出视图及视图结构，并且语法格式也相同。

1) 查看视图列表

例如下述命令可以查看表和视图的列表，结果如图 6-19 所示。

```
Show tables;
```

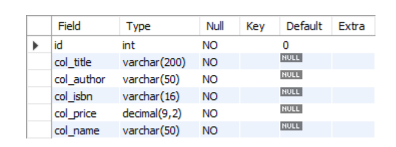
2) 查看视图的数据结构

使用下述命令查看视图（或表）的数据结构。

```
Describe 表名或视图名;
```

例如查看视图 v_book 的结构，结果如图 6-20 所示。

```
Describe v_book;
```



Field	Type	Null	Key	Default	Extra
id	int	NO		0	
col_title	varchar(200)	NO		NULL	
col_author	varchar(50)	NO		NULL	
col_isbn	varchar(16)	NO		NULL	
col_price	decimal(9,2)	NO		NULL	
col_name	varchar(50)	NO		NULL	

图 6-20 视图 v_book 的结构

3) 列出创建视图的 Create view 语句

如果要查看创建视图所使用的 Create view 语句的代码，可以使用下述语句。

```
Show create view v_book;
```

参考单元 3 的“3.3.5 列出表及表结构”，只是这里用视图替换了表。

2. 变更视图

可以使用 Alter view 语句变更一个已经存在的视图，其语法格式与 Create view 相同，不同的是如果原视图不存在，将会出错。

3. 丢弃视图

视图是一种数据库对象，视图被创建后将作为数据结构的一部分而永久存在，除非将其丢弃。丢弃视图与丢弃表的语法格式相同，如下所示。

```
Drop view [if exists] <视图名>;
```

6.3 索引

索引可以极大地提高查询的速度，就像在新华字典中查找一个字的读音，如果没有索引，就需要从第一页翻到最后一页，一个字一个字地找，而通过笔划索引或偏旁索引，则可以很快找到这个字。

索引的优点如下。

- 提高查询数据（where 子句）的速度。
- 通过唯一性索引，可以实现唯一性约束
- 提高实现外键约束、分组查询和排序子句的速度

索引的缺点如下。

- 索引的创建和维护需要耗费 CPU 时间，会降低插入、更新、删除的速度。
- 索引本身需要占用磁盘空间，消耗计算机资源。

6.3.1 索引的分类

1. 按逻辑功能划分

- 普通索引：为提高查询性能而创建的索引。
- 唯一索引：为实现唯一性约束而创建的索引，它也具有普通索引提高查询性能的作用。
- 主键索引：为主键创建的索引，它是唯一索引，效率是最高的。
- 全文索引：是用于对文本列进行全文搜索的索引，针对全文搜索的特点进行了优化。

2. 按物理实现划分

- 聚簇索引：索引和数据存储合二为一，因此索引的顺序与数据的物理保存顺序是相同的，聚簇索引只能有一个。如同新华字典，每个字都是按拼音排列的，这个拼音索引就是聚簇索引。这种索引效率最高，通常主键索引就是聚簇索引。
- 非聚簇索引：索引和数据分开存储在磁盘上，效率低一些。除聚簇索引外的其他索引都是非聚簇索引。如同新华字典的笔划索引或偏旁索引。

3. 按列的个数划分

单列索引：仅对一列进行的索引。

组合索引：对多列进行的索引，可以是普通索引，也可以是唯一性索引。

4. 按索引算法区分

可以分为 B+tree 索引、Hash 索引、Fulltext 索引等。

6.3.2 索引的设计原则

1. 应该建立索引的情形

在下列情况下需要建立索引，通常根据业务需求来决定哪些列需要建立索引，以及索引的类型。

- 主键必须建立索引，这是默认的和强制性的，这种索引是唯一性索引，也是聚簇索引。
- 不允许出现重复值的列或多个列（即候选键），这时需要建立唯一性索引（单列索引或组合索引）。
- 经常查询的列应该建立索引，这时需要建立普通索引，有时是组合索引。
- 外键、排序的列（order by）、分组统计的列（group by）应该建立索引。

2. 不应该建立索引的情形

在下列情况下不应该建立索引，主要考虑的因素是在这种情况下建立索引并不能有效的提高效率。

- 从来不查询或很少查询的列不应建立索引，例如备注列。
- 对于行数少的表不需要建立索引，例如全国省份表，只有 34 行。

- 对于取值范围很小的列不应建立索引，例如性别列，只有“男”和“女”两种值。

6.3.3 创建索引

1. 间接创建索引



在创建表时，通过定义列的主键、唯一性约束等可以同时创建索引。例如下述创建表的语句，同时创建了两个唯一性索引。

```
Use abc;
Drop table if exists publisher;              -- 丢弃表的同时将丢弃索引
Create table publisher (
	id int primary key not null auto_increment,
	col_name varchar(50) unique null,
	col_addr varchar(50)
);
```

其中 primary key 创建一个主键约束，它具有唯一性约束，也是唯一性索引。而关键字 unique 为 col_name 创建一个唯一性约束，也是唯一性索引。

2. 直接创建索引

1) 使用 Create index 语句

通过 Create index 语句创建，在创建表之后可以添加索引，语法格式如下。

```
Create [unique] index <索引名> on <表名(列名 1, 列名 2, …)>;
```

例如对于出版社表和图书表，调查发现，用户对书名的查询比较频繁，因此需要为图书表（book）的书名列（col_title）添加一个普通索引，索引名为 idx_book_col_title，从而加快查询书名的性能。

```
Create index idx_book_col_title on book(col_title);
```

2) 使用 Alter table 语句

也可以通过为表添加一个索引来实现，语法格式如下。

```
Alter table 表名
	add [unique] index 索引名(列名 1, 列名 2, …);
```

与上述 Create index 语句等价的代码如下所示。

```
Alter table book
	add index idx_book_col_title(col_title);
```

3. 创建组合索引

有时需要创建由多列组成的组合索引，例如对于下述成绩表。

```
Create table score (
	id int primary key not null auto_increment,
	col_score tinyint not null default '0',          -- 成绩
	col_term varchar(50) not null default '',        -- 学期
	id_student int not null default '0',             -- 学生 ID
	id_course int not null default '0'               -- 课程 ID
);
```

要唯一标识一个成绩，需要“学生 ID+课程 ID+学期”的组合是唯一的，因此要创建一个唯一性的组合索引，代码如下。

```
Create unique index udx_score_student_course_term on score(id_student, id_course, col_term);
```

索引名以 udx 起头，表示是唯一性索引，索引名包含了 3 个列名，表示这 3 个列的组合索引。

6.3.4 使用索引

索引不是直接使用的，它是提高数据库性能的极其重要的手段，能够成倍甚至几万倍地提高查询性能。



索引的作用是提高数据库查询的性能，在单元 9 的“9.6 性能优化与索引”中，在性能优化方面如何使用索引作了较为深入的讲解。

6.3.5 管理索引

索引是一种数据库对象，因此创建索引后，可以在导航栏看到创建的索引，如图 6-21 所示。在表设计界面中单击 Indexes 选项卡，也可以看到索引的列表，如图 6-22 所示。

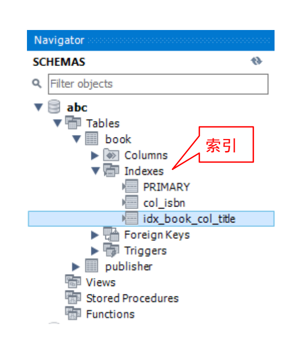


图 6-21 导航栏中的索引列表

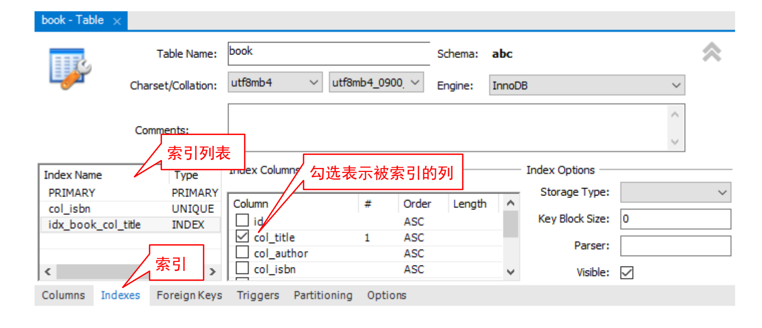


图 6-22 表设计界面中的索引列表

1) 列出索引列表

例如下述语句列出 book 表的索引列表，如图 6-23 所示。

```
Show keys from book;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
book	0	PRIMARY	1	id	A	9	NULL	NULL		BTREE			YES	NULL
book	0	col_isbn	1	col_isbn	A	9	NULL	NULL		BTREE			YES	NULL
book	1	idx_book_col_title	1	col_title	A	9	NULL	NULL		BTREE			YES	NULL

图 6-23 book 表的索引列表

图 6-23 显示 book 表有 3 个索引，其中有两个唯一性索引（Non_unique 列的值为 0），即主键 id 的唯一性索引和 col_isbn（书号）的唯一性索引，一个非唯一性索引（Non_unique 列的值为 1），即刚刚创建的普通索引。

2) 丢弃索引

可以直接丢弃一个索引，也可以通过修改表来丢弃表中的索引。例如下述两条语句都可以丢弃 book 表上名为 idx_book_col_title 的索引。

```
Drop index idx_book_col_title on book;
```

或者下述语句具有相同的结果。

```
Alter table book
    drop index idx_book_col_title;
```

3) 修改索引

索引无法修改，因此 MySQL 不提供修改索引的语句。需要时应先丢弃索引，再重新创建索引。

【案例讲解】书店管理系统的子查询、视图和索引

打开附录 D 的“项目 2c 书店管理项目”，看看在项目中的哪些地方需要使用子查询、哪些地方可以使用视图、以及可以建立哪些索引。



任务 1 使用子查询

在“项目 2c 书店管理项目”中的订单录入模块中，不论是新增订单或编辑订单，在保存订单数据时，都需要更新订单的“总金额”列（col_total_ammount），这是通过在“保存数据的 SQL 语句”中增加两条

更新语句来实现的，代码如下（加粗斜体的部分）。

```
-- 包含主表和从表的代码如下：
E1: Begin
Start transaction;
Call p_jitor_saver(@err, 'bs_order', '{$rowData}', @id, '', ''); -- 第二个参数是 bs_order，最后两个参数必须为空串，其中@id 传主表的主键值给从表
If @err is not null then
  Select '保存主表时出错（回滚）' msg, @err error;   -- 返回错误信息
    -- 存储过程中已经回滚
    Leave E1;
End if;
Call p_jitor_saver(@err, 'bs_order_detail', '{$childData}', @id, 'id_bs_order', 'bs_order'); -- 第二个参数是 bs_order_detail，最后两个参数指定参照关系（注意检查外键的名称是否正确）
If @err is not null then
  Select '保存从表时出错（回滚）' msg, @err error;   -- 返回错误信息
    -- 存储过程中已经回滚
    Leave E1;
End if;
-- 需要时，可以回滚，并返回出错信息：Select '自定义出错信息' msg, 'E98009' error;
Update bs_order_detail set col_ammount = col_price * col_quantity;      -- 更新订单明细表的金额列
Update bs_order set col_total_ammount =                                 -- 更新订单表的总金额列
    (Select sum(col_ammount)                 -- 使用子查询，只更新当前订单（@id 的这一行）
        from bs_order_detail
        where id_bs_order =@id
    ) where id = @id;
Commit;
Select '成功保存主表和从表' msg, '' error;
End;
```

任务 2 使用视图

在实战演练平台上，所有使用 Select 的地方，都可以预先创建视图，然后在查询语句中使用视图，提高代码的复用。

常常是将复杂的，并且需要复用的语句改为视图，例如“按出版社统计”的主表数据源语句比较复杂，如图 6-24 所示，代码如下。

```
Select bs_publisher.col_name name, sum(col_quantity) quantity, sum(col_amount) amount
    from bs_order_detail
        join bs_book on bs_order_detail.id_bs_book = bs_book.id
        join bs_publisher on bs_book.id_bs_publisher = bs_publisher.id
        join bs_order on bs_order_detail.id_bs_order = bs_order.id
    where bs_order.col_status = '已发货'
    group by bs_publisher.id;
```

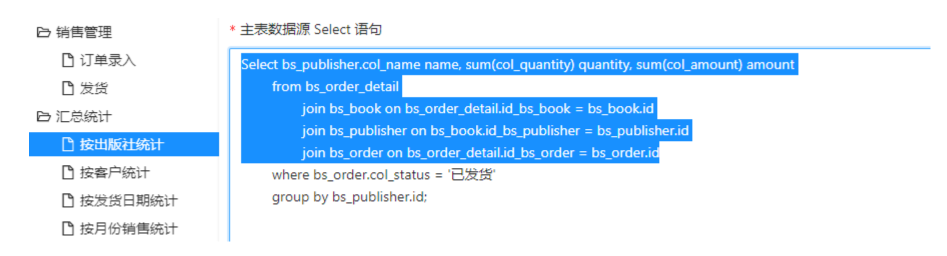


图 6-24 将用于创建视图的 Select 语句的一部分

这时将语句的内连接部分创建为一个名为 v_sales_by_publisher 的视图，代码如下。

```
Create view v_sales_by_publisher as
Select bs_publisher.id, bs_publisher.col_name, col_status, col_quantity, col_amount
    from bs_order_detail
```

```
            join bs_book on bs_order_detail.id_bs_book = bs_book.id
            join bs_publisher on bs_book.id_bs_publisher = bs_publisher.id
            join bs_order on bs_order_detail.id_bs_order = bs_order.id;
```

创建视图后，主表数据源语句就可以写为如下。

```
Select col_name name, sum(col_quantity) quantity, sum(col_amount) amount
      from v_sales_by_publisher
      where col_status = '已发货'
      group by id;
```

这时，如果另外一个模块要求统计“未发货”的数据，则可以用下述语句作为主表数据源语句。

```
Select col_name name, sum(col_quantity) quantity, sum(col_amount) amount
      from v_sales_by_publisher
      where col_status <> '已发货'          -- 等于号改为不等于
      group by id;
```

从而简化了代码的编写，提高了代码的复用。

任务 3 使用索引

为了实现唯一性约束或提高查询性能，需要为表的某些列创建索引，实现唯一约束，或提高查询效率。例如为出版社表的出版社名称建立唯一性索引，避免出版社名称的重复，或者为图书表的书名建立普通索引，提高查询性能。

【实战演练】图书借阅系统的子查询、视图和索引



附录 D 项目 2d

参考附录 D 的“项目 2d 图书借阅项目”，在读者自行开发的图书借阅项目上，根据项目的需求，灵活运用子查询和视图来编写各种查询语句。根据索引的设计原则，对经常查询的列、参与排序的列、进行分组统计的列创建索引。

【单元重点】

单元小结参见本单元的【思维导图】，单元重点如下。

- 简单子查询和相关子查询各自的特点，以及它们的区别。
- 子查询与增删改语句的联合使用。
- 视图的作用和特点。
- 索引的作用、分类和设计原则。

【课后思考】

四、选择题

1. Select 查询的结果从本质上看是（　　）。

   A. 一张表　　B. 一个集合　　C. 一个值　　D. 以上都是

2. Select 查询的结果可以用在使用（　　）的位置。

   A. 一张表　　B. 一个集合　　C. 一个值　　D. 以上都是

3. 子查询作为一张表使用时，必须【　　】。

   A. 用圆括号括起来　　B. 指定一个别名　　C. 可以使用表的位置　　D. 返回多行多列

4. 子查询作为一个集合使用时，必须【　　】。

   A. 用圆括号括起来　　B. 指定一个别名　　C. 可以使用集合的位置　D. 返回多行一列

5. 子查询作为一个值使用时，必须【　　】。

   A. 用圆括号括起来　　B. 指定一个别名　　C. 可以使用值的位置　　D. 返回一行一列

6. In 子查询是将子查询作为一个（　　）作用的。

A. 一张表　B. 一个集合　C. 一个值　D. 一个逻辑值

7. Exists 子查询是将子查询作为一个（　　）作用的。

A. 一张表　B. 一个集合　C. 一个值　D. 一个逻辑值

8. 简单子查询的特点是【　　　】。

A. 子查询需要引用父查询的表　B. 子查询本身可以独立执行

C. 先执行子查询，然后再执行父查询　D. 一共执行 1+N 次查询（N 是父查询行数）

9. 相关子查询的特点是【　　　】。

A. 子查询需要引用父查询的表　B. 子查询本身可以独立执行

C. 先执行子查询，然后再执行父查询　D. 一共执行 1+N 次查询（N 是父查询行数）

10. 子查询可以与【　　　】等语句协同使用。

A. Create table　B. Insert　C. Update　D. Delete

五、填空题

1. Create table book2 as Select * from book where 1=2; 语句的作用是____________________________。

2. Insert into book2 Select * from book; 语句的作用是____________________________。

六、思考题

1. 简单子查询与相关子查询有哪些异同点？

2. 复制表（Create table … like …）和保存查询结果为一张表（Create table … as …）有什么不同？

【课外拓展】

1、 阅读一篇介绍视图的文章，增进对视图的理解。

2、 阅读一篇介绍索引的文章，充分理解索引的作用。

单元7 数据库编程

【学习目标】

知识目标	能力目标
◆ 了解存储程序的种类和优缺点。	◆ 学会语句分隔符的设置。
◆ 掌握 MySQL 编程知识。	◆ 学会使用内置函数。
◆ 熟练掌握常用的内置函数。	◆ 初步学会创建和使用存储函数和存储过程。
◆ 理解存储函数、存储过程和触发器的异同点。	◆ 初步学会创建和使用触发器和事件。
◆ 初步了解事件的作用。	◆ 学会事务的开启、提交和回滚语句的使用。
◆ 深刻理解事务，特别是事务的 ACID 特性。	**素质目标**
◆ 理解并发访问时的问题。	◆ 多用户并发环境、团队精神、诚信与责任。
◆ 初步了解锁的作用。	◆ 安全意识、诚信与法治。

【思维导图】



【情景导入】

小明学完了单元 6，觉得 SQL 功能非常强大，可以满足各种各样的需求。他从学长那里还得知，SQL 还能够像 C++或 Java 那样编程，提供更加强大的的功能。小明感到这是一个挑战，迫切地想知道如何进行 SQL 编程，让我们同小明一起探索数据库编程吧。

【知识储备】

数据库是保存数据库对象的容器，这些数据库对象分为两大类，一类用于保存数据（表，包括表结构和表中的数据），另一类用于保存 SQL 语句（视图、存储函数、存储过程、触发器和事件）。

单元 3～单元 5 主要讲解了表，包括表的创建、操纵和查询，单元 6 讲解了子查询和视图，还讲解了与表有关的索引。本单元主要讲解存储函数、存储过程、触发器和事件，并讲解事务和锁的概念。

7.1 MySQL 编程

7.1.1 存储程序和存储例程

数据库编程时编写的代码都要保存在存储程序中，因此先了解一下存储程序有哪些种类。

1. 存储程序

存储程序（stored programs）是存储在数据库中的 SQL 代码，由一行语句或多行语句组成，并给予一个命名，通过该名字来运行或管理这些语句。MySQL 支持下述四种存储程序。

- 存储函数（stored function：它返回一个计算结果，该结果用在表达式里（例如 Select 语句中的计算列）。
- 存储过程（stored procedure）：它不直接返回一个结果，但可以用来完成一般的运算或是用 select 语句生成一个结果集并传递回调用方，它被 call 命令调用。
- 触发器（trigger）：它与表相关联，不能被直接运行，而是在该表执行 insert、delete 或 update 语句时触发它的执行。
- 事件（event）：它也不能被直接运行，根据设置的时间，在设置的预定时刻自动执行。

2. 存储例程

存储例程（stored routine）仅指存储函数和存储过程两种。之所以将存储函数和存储过程单独归类于存储例程，是由于在数据库备份时，有一个例程选项，用于指定是否备份存储函数和存储过程

3. 存储程序的优缺点

1) 存储程序的优点

存储程序曾经受到广泛的应用，有些项目甚至将主要或所有的业务逻辑都写在各种存储程序中。存储程序的优点如下所示。

- 编译后执行：存储程序编译后的执行速度更快，从而提高性能。
- 减少网络传输：存储程序保存在 MySQL 服务器中，减少网络传输的开销，从而提高效率。
- 代码复用：存储程序可以被不同的进程甚至是不同的语言调用，从而提高开发效率。
- 流程控制：可以使用流程程控语句，实现复杂的判断和运算，编写出比较通用的存储程序。
- 安全性和完整性：存储程序是一个数据库对象，每个存储程序是一个完整的业务逻辑，并能够进行权限控制。

2) 存储程序的缺点

存储程序也有其重大的缺点。存储程序的缺点如下所示。

- 维护困难：将核心业务的代码集中在存储程序中，很容易使代码膨胀到上千行代码，修改很少的代码就可能影响到许多业务功能。
- 调试不便：不提供调试手段，几乎无法调试。
- 移植性差：缺少标准，在不同的 DBMS 之间移植相当困难，几乎是要完全重写。

● 降低效率：占用服务器端太多的资源，对服务器造成很大的压力。

总之，存储程序是有利有弊，需要根据实际情况选择使用，一般来说，对于小型项目，存储程序还是利大于弊，而对于大型项目，则弊大于利，以致于现在互联网大厂强烈反对使用存储程序，特别是严格禁止使用触发器等。



不建议或禁止使用存储程序并不代表存储程序不重要，而是权衡利弊之后作出的无奈之举。这也意味着存储程序的功能要由程序员在应用程序层面上加以实现，这也是我们仍然要学习存储程序的原因之一。

7.1.2 语句块和语句分隔符

在 MySQL 中编写存储程序有一个特殊之处，那就是对行末分号的处理。

如果组成存储程序的语句只有一条，则不需要作特别的处理。例如下述代码创建一个单语句的存储程序（以存储过程为例）。

```
Create procedure  存储过程名(参数)
        一条语句;
```

如果组成存储程序的语句有多条，就需要用 Begin 和 End 关键字将多条语句括起来，成为一个语句块。例如下述代码创建一个多语句的存储程序（以存储过程为例）。

```
Create procedure  存储过程名(参数)
Begin
        第一条语句;
        ……
        第 n 条语句;      -- 存储程序内的行末分号
End;                      -- 存储程序外的行末分号，两者相同时，无法区分
```

因为语句块中的代码本身含有行末分号，因此需要作特别的处理。这时为了区分存储程序内的行末分号和存储程序外的行末分号，MySQL 要求指定一个新的分隔符，以区分它们。例如下述命令指定双美元号为新的分隔符。

```
Delimiter $$
```

这时存储程序内使用分号作为分隔符，存储程序外用双美元号作为分隔符，直到重新指定分号为分隔符（这里分号前应该有一个空格，分号作为 delimiter 命令的参数）。

```
Delimiter ;
```



Delimiter 是一条极为特殊的 MySQL 命令，它后面的任何内容都是参数，参数只有两种：新的分隔符（如$$）或分号（;），并且在 Delimiter 和作为参数的符号之间要加上空格加以分隔。

因此，编写一个存储程序的格式如下所示。

```
Delimiter $$              -- 指定新的分隔符为$$（本行不能添加注释）
Create {procedure|function|trigger|event}  存储程序名[(参数)]
Begin
        第一条语句;
        ……
        第 n 条语句;      -- 存储程序内的行结束符（分号）
End$$                     -- 存储程序外的行结束符（双美元号），这时两者不同
Delimiter ;               -- 恢复分号为默认的分隔符（分号是 Delimiter 命令的参数，要有空格分隔）
```



Delimiter 命令的行末不能加注释，如果加了注释，则注释也成为参数的一部分，而导致无法执行，并且不会有错误信息的提示。上述代码中 Delimiter 命令含有注释就是错误的。

下述代码是一个可以运行的存储过程，本单元多数例子代码都要按如下格式写在语句块中。



附录 C 实训 7-1

```
Drop procedure if exists p_proc;
Delimiter $$
Create procedure p_proc()
Begin
	Select 3*4  计算结果;
	-- 更多语句代码;
End$$
Delimiter ;
```

对于存储过程来说，创建好之后，要 call 关键字调用，调用前述 p_proc 存储过程的代码如下。

```
Call p_proc();
```

7.1.3 图形界面编写存储例程

上一小节使用语句分隔符是十分不便的，MySQL Workbench 提供了图形界面来创建存储例程（存储过程或存储函数），避免了使用语句分隔符的不便，如图 7-1 所示。

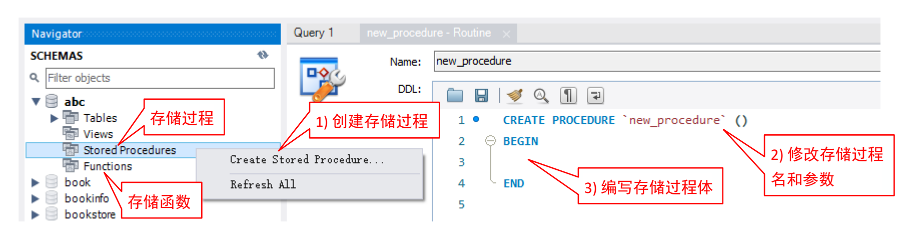


图 7-1 图形界面编写存储过程

编写好后，单击 Apply 按钮执行，在生成的代码中，自动加上分隔符的处理，如下所示。

```
USE `abc`;
DROP procedure IF EXISTS `new_procedure`;

DELIMITER $$
USE `abc`$$
CREATE PROCEDURE `new_procedure` ()
BEGIN
	存储过程体的代码;
END$$

DELIMITER ;
```

7.1.4 编程基础

到目前为止，编写的 SQL 语句都是独立执行的，SQL 语言也支持编程语言的功能，拥有分支、循环等程序结构，用以编写存储函数、存储过程或触发器等存储程序。

编程语言的基础是关键字、数据类型、变量和常量、运算符、表达式，以及流程控制，下面分别讲解。

1. 关键字

MySQL 的关键字有近 400 个，另外还有 200 多个保留字（保留用于今后作为关键字使用），共计 600 多个，这些关键字和保留字虽然可以作为表名或列名使用，但是有一定的附加要求，即需要用反引号（位于键盘 TAB 键的上方）括起来，例如 order 是关键字，作为列名时要写成`order`，因此不建议使用关键字作为表名、列名等使用。

建议所有命名采用小写，但对数据库名和表名等，特别强调采用小写，因为在 Linux 平台，这些名字是区分大小写的，而在 Windows 平台则不区分大小写。

2. 数据类型

变量的数据类型与列的数据类型相同，见附录 A，并参见单元 3“3.3.1 数据类型”。

3. 变量

MySQL 的变量有三种，如表 7-1 所示。

表 7-1 变量的类型

	系统变量	用户变量	局部变量
命名	强制规定以@@作为前缀	强制规定以@作为前缀	建议用 var_作为前缀
用途	系统定义的变量，可能影响全局	用户定义的变量，保存数据	在语句块中定义的临时变量
定义	系统预定义，不允许自定义	用 set 定义	必须用 declare 声明
赋值	用 set 赋值（只读的不能赋值）	用 set 或 select 赋值	用 set 赋值或 select 赋值
查看	select 语句	select 语句	select 语句
作用域	全局	同一个连接	语句块中

这三种变量的最大区别是用途的不同，因此需要根据用途来选择合适的变量类型。

1) 系统变量

系统变量是 MySQL 系统内部定义的变量，保存了系统的配置参数，以及软件和硬件参数（操作系统类型、CPU 类型等）。例如下述代码查询系统变量@@version 的值，它是 MySQL 的版本号。

```
Select @@version;
```

2) 用户变量

用户变量是用户定义的变量，用于保存数据，具有较长的生命周期。例如下述代码。

```
Set @text = "Hello";            -- 赋值
Select @text;                   -- 用 select 语句输出值
```

还可以用 select 语句加上 into 关键字来赋值，例如下述代码将查询结果赋给两个变量@ammout 和变量@average。

```
Select sum(col_quantity), avg(col_price)        -- 创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”
    into @quantity, @average from book;
Select @quantity, @average;
```

3) 局部变量

局部变量是在语句块（用 begin 和 end 定义语句块）中声明的变量，用于保存临时数据，它的生命周期很短，并且需要声明它的数据类型。

局部变量必须先用 declare 声明，然后才能使用，语法格式如下。

```
Declare 局部变量名 数据类型 default 初始值;
```

例如下述代码中的局部变量 var_text，它只在语句块中有效，因为语句块必须存在于存储程序中，这里用存储过程来演示。

```
Drop procedure if exists p_proc;
-- 下一行语句参见“7.1.2 语句块和语句分隔符”
Delimiter $$
Create procedure p_proc()
Begin
    Declare var_text varchar(50);       -- 声明局部变量时，可以指定初始值
    Set var_text = "Hello";             -- 赋值
    Select var_text;                    -- 用 select 语句输出值
End$$
Delimiter ;
```

上述代码创建一个名为 p_proc 的存储过程，这个存储过程只有 3 行代码，没有实质性功能，仅是演示局部变量的使用。

调用这个存储过程的语句如下，运行的结果是显示局部变量的值，如图 7-2 所示。

```
Call p_proc();
```

var_text
Hello

图 7-2 局部变量 var_text 的值

4. 变量的赋值

变量的赋值可以用 set 或 select 语句，根据应用场景选择使用。

1) Set

使用 set 语句将一个值直接赋给变量，例如下述代码。

```
Set @a = 3;                        -- 直接赋值
Set @a := 3;                       -- 直接赋值
```

其中赋值符可以是“=”，也可以是“:=”，效果是相同的。

2) Select

使用 select 语句可以将一个值或表中列的值赋给变量，例如下述代码。

```
Select @a := 3;                    -- 直接赋值
Select @a := 3, @b := 'abc';       -- 赋值给多个变量
```

其中赋值符只能是“:=”，而不能是“=”，否则赋值是失败的（但不出错）。

还可以用 into 关键字来赋值，与前者的区别是不会在输出区显示 select 语句的结果。

```
Select 3 into @a;                  -- 用 into 关键字赋值
Select 3, 'abc' into @a, @b;       -- 赋值给多个变量
```

Select 还可以将查询得到的值赋值给变量，而 set 则不行。

```
Select @a := sum(列名) from  表名;          -- 将查询结果赋给变量，要求返回零行或一行，显示结果。
Select sum(列名) into @a from  表名;        -- 将查询结果赋给变量，要求返回零行或一行，不显示结果。
```

5. 字面常量

字面常量是直接用文字表示的固定不变的数据，常用的字面常量有整数、浮点数、字符串和日期等，如 12、1.23 和"Hello"，如表 7-2 所示，详见单元 3“3.3.1 数据类型”对字面常量的讲解。

表 7-2 常量的表示

常量	说明	例子
整数值	十进制整数	12
浮点数值	带小数的十进制数，也可以用科学记数法表示	3.14、5.1E5（表示5.1×10^5）
字符串值	用单引号或双引号括起来的 0 到多个字符，可用转义字符	"It's me."、'It\'s me.'
日期时间值	用单引号括起来的表示日期、时间或日期时间的字符串	'2020-05-15'、'2020-05-15 16:42:02'
布尔值	只有 TRUE 和 FALSE 两个值，分别表示真和假	True
NULL 值	只有 NULL 一个值，表示空	Null

6. 运算符和表达式

常用运算符在单元 5“5.1.3 选择行 Where”一节中讲解过。

表达式是以一定的运算规则，用运算符将常量、变量以及标识符等连接而成的算式，表达式可以用在 Select 语句中作为计算列，也可以用在其他需要值的地方。

7.1.5 流程控制语句



附录 C 实训 7-2

流程控制语句有分支和循环两大类。流程控制语句只能用在存储程序中，而不能单独执行。必须将流程控制语句放在语句块中（存储程序框架代码可从“项目 3a 公共代码共享”复制），然后通过“call 存储过程名”来执行，。

1. 条件分支语句

MySQL 支持两种条件分支语句：If 条件分支语句和 case 条件分支语句。

1) If 条件分支语句

If 条件分支采用 if…elseif…else…end if 的结构，例如下述语句。

```
Set @id=3;          -- 存储程序框架代码见“项目 3a 公共代码共享”
If @id=1 then
        select "语句 1";
Elseif @id=2 then
        select "语句 2";
Else
        select "语句 n";
End if;
```

2) Case 条件分支语句

也可以采用 case 语句实现条件分支，例如下述代码与前述 if 条件分支的作用相同。

```
Set @id=2;
Case @id
when 1 then
        select "语句 1";
when 2 then
        select "语句 2";
else
        select "语句 n";
End case;
```

上述代码也可以写为如下代码。

```
Set @id=2;
Case
when @id=1 then
        select "语句 1";
when @id=2 then
        select "语句 2";
else
        select "语句 n";
End case;
```



if 语句、case 语句与“5.1.2 选择列”最后一部分讲解的 if()函数、case 运算符是不同的，前者用于流程控制，后者用于表达式。

2. 循环语句

MySQL 支持三种循环语句：While 循环语句、Repeat 循环语句和 Loop 循环语句。

1) While 循环语句

While 循环语句的语法格式如下。

```
[标签:] While 循环条件表达式 do
    语句块;
End while;
```

可选地，可以用 leave 关键字强制退出当前循环，或用 iterate 关键字强制进入下一次循环。



与 C/C++、Java 语言比较，两者的 while 循环是类似的，leave 相当于是 break，iterate 相当于是 continue。而后面讲解的 repeat 循环则与 do…while 循环类似。

下面是一个计算 1 到 100 的累加和的例子。

```
Declare _i int default 0;          -- 存储程序框架代码见“项目 3a 公共代码共享”
Set @sum = 0;
```

```
    While _i<100 do
        Set _i = _i + 1;
        Set @sum = @sum + _i;
    End while;
    Select @sum;
```

下述代码加了一个 iterate 部分，在某些条件下跳过了循环体后面的语句，因此只计算 1 到 50 的累加和（虽然循环条件是到 100）。

```
    Declare _i int default 0;
    Set @sum = 0;
r:  While _i<100 do -- 用标签 r 命名这个循环
        Set _i = _i + 1;
        If _i>50 then
            iterate r; -- 跳过后面的循环体，进入 r 指定的循环的下一次循环
        End if;
        Set @sum = @sum + _i;
    End while;
    Select @sum;
```

2) Repeat 循环语句

Repeat 循环语句的语法格式如下。

```
[标签:] Repeat
    语句块;
    Until 结束条件表达式
End repeat;
```

下面是一个计算 1 到 100 的累加和的例子。

```
    Declare _i int default 0;
    Set @sum = 0;
    Repeat
        Set _i = _i + 1;
        Set @sum = @sum + _i;
        Until _i>=100    -- 到达 100 后，结束循环
    End repeat;
    Select @sum;
```

3) Loop 循环语句

Loop 循环语句的语法格式如下。

```
标签: Loop
    语句块;
End loop;
```

从这个格式中看到，loop 循环没有结束条件，因此它是无限循环的，需要 leave 语句的帮助才能结束。因此，标签是必须的，否则无法结束循环。

下面同样是一个计算 1 到 100 的累加和的例子。

```
    Declare _i int default 0;
    Set @sum = 0;
r:  Loop -- 用标签 r 命名这个循环
        Set _i = _i + 1;
        Set @sum = @sum + _i;
        If _i>=100 then
            leave r; -- 退出标签 r 指定的循环（否则无限循环）
        End if;
    End loop;
    Select @sum;
```

这三种循环各有其特点，如表 7-3 所示。

表 7-3 三种循环语句的比较

比较项	while 循环	repeat 循环	loop 循环
循环类型	当型循环	直到型循环	无限循环
条件表达式	循环条件，为真时继续循环	结束条件，为真时结束循环	无，需要 leave 语句来结束
特点	可以循环 0 次到多次	循环 1 次到多次，至少 1 次	取决于 leave 语句
与 C/Java 比较	while 循环	do…while 循环	例如 for(;;)或 while(1)

7.2 内置函数



内置函数是 MySQL 本身提供的，可以被直接调用。MySQL 函数可以用在表达式中，包括 select 语句中的计算表达式。MySQL 提供了丰富的内置函数，涵盖了编程的各种需要。更多的内置函数见附录 B。

1. 聚合函数

在“5.4 聚合查询”一节讲解过，不再赘述。

2. 数学函数

数学函数包括常用的数学计算，例如下述语句对 3.14159 进行四舍五入（保留 3 位小数）。

```
Select round(3.14159, 3);
```

3. 字符串函数

字符串函数用于对字符串进行处理，常用的字符串函数及其例子如表 7-4 所示。

表 7-4 常用的字符串函数

函数名	功能	例子	结果
length(string)	返回字符串的字节数	length('汉字 abc');	9（一个汉字占 3 个字节）
char_length(string)	返回字符串的字符数	char_length('汉字 abc');	5（一个汉字就是一个字符）
substring(string, start, length)	求字符串的子串	substring('12345678', 2, 3)	'234'
replace(string, s1, s2)	替换字符串	replace('12345678', '23', 'abc')	'1abc45678'
concat(string1, string2, …)	连接字符串	concat('12345678', '23', 'abc')	'1234567823abc'
ascii(character)	求字符的 ASCII 值	ascii('abc')	97（只返回第一个字符的 ASCII 值）
char(integer)	从 ASCII 值得到字符	char(97)	'a'

使用字符串函数的例子如下。

```
Select substring('12345678', 2, 3);          -- 结果是 234
```

4. 日期和时间函数

日期和时间函数包括返回当前的日期时间、对日期时间进行加减等。

例如下述语句获得当前日期的月份值。

```
Select month(now());
```

例如下述两条语句分别获得指定日期加上 50 天的日期，以及减去 500 小时的日期时间。

```
Select adddate('2025-01-01', interval 50 day);          -- 结果是 '2025-02-20'
Select subdate('2025-01-01', interval 500 hour);        -- 结果是 '2024-12-11 04:00:00'
```

5. 系统函数

1) MySQL 版本号

系统函数与 MySQL 服务器有关，例如可以从 version()函数获得 MySQL 版本号。

```
Select version();
```

2) 自动生成的主键值

如果主键是自增量的，有时需要知道新插入行的自动生成的主键值是多少。这里以单元 5 的 demo 表为例（参见图 5-1），代码如下。

```
Insert into demo (name) values ('xxx');  -- 创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”
Select last_insert_id() into @id;
Select @id;
```

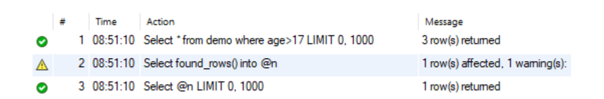
结果是将新插入行自动生成的 id 值赋给变量@id，以便后续的处理。

3) 查询返回的行数

对表进行查询，然后可以通过 found_rows()函数取得这个查询找到了多少行。

```
Select * from demo where age>17;        -- 创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”
Select found_rows() into @n;
Select @n;
```

结果是将 found_rows()赋给变量@n，这个例子的值是 3，也就是如图 7-3 所示的返回 3 行。



#	Time	Action	Message
1	08:51:10	Select * from demo where age>17 LIMIT 0, 1000	3 row(s) returned
2	08:51:10	Select found_rows() into @n	1 row(s) affected, 1 warning(s):
3	08:51:10	Select @n LIMIT 0, 1000	1 row(s) returned

图 7-3 found_rows()值是返回的行数

4) 与更新有关的行数

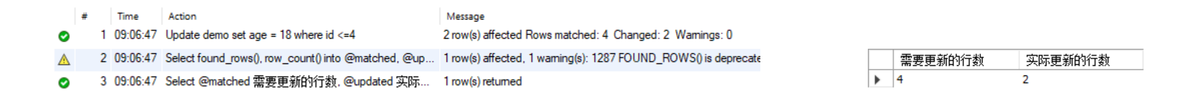
对于更新操作，则情况比较特殊。与更新操作的行数有关的指标有如下两个。

- 需要更新的行数：与 update 语句的 where 条件有关，这个值通过 found_rows()函数取得。
- 实际更新的行数：如果新值与原值相同，则无需更新，通过 row_count()函数取得的值是实际更新的行数。

例如下述代码将 demo 表中 id 小于等于 4 的 age 更新为 18。

```
-- 创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”，需要时可再次初始化
Update demo set age = 18 where id <=4;
Select found_rows(), row_count() into @matched, @updated;
Select @matched 需要更新的行数, @updated 实际更新的行数;
```

表中共有 5 行或更多行，满足 where 条件的只有 4 行，其中有 2 行的 age 已经是 18，无需更新，实际更新了其余 2 行。图 7-4 是在信息显示区显示的信息，图 7-5 则是在代码中取得的这两个值。



#	Time	Action	Message
1	09:06:47	Update demo set age = 18 where id <=4	2 row(s) affected Rows matched: 4 Changed: 2 Warnings: 0
2	09:06:47	Select found_rows(), row_count() into @matched, @up...	1 row(s) affected, 1 warning(s): 1287 FOUND_ROWS() is deprecate
3	09:06:47	Select @matched 需要更新的行数, @updated 实际...	1 row(s) returned

需要更新的行数	实际更新的行数
4	2

图 7-4 更新操作的信息显示　　　　图 7-5 与更新操作有关的行数

5) 删除的行数

对表进行删除，有可能删除一行或多行，这时可以用 row_count()函数取得实际删除的行数。

```
Delete from demo where weight is null;
Select row_count();
```

6. 转换函数

转换函数将一种数据类型的值转换为另一种数据类型的值，例如下述两条语句的作用相同，都是将字符串转换为数字。

```
Select cast('1.236' as decimal(5,2));        -- 结果是 1.24（四舍五入，精确浮点数）
Select convert('1.236', decimal(5,2));       -- 结果是 1.24（四舍五入，精确浮点数）
```

Cast()和 convert()在功能上几乎完全相同，不同的是前者是 SQL 标准，后者则是兼容的。

7.3 存储函数

上一节讲解了内置函数，MySQL 支持自定义函数，这种自定义函数称为存储函数。在实际项目中，存储函数是比较常见的一种存储程序。

7.3.1 存储函数的创建和使用

创建存储函数的语法格式如下。

```
Create function 存储函数名称(参数列表)
	returns 返回值类型 [存储函数特征]
	存储函数体
```

参数说明如下。

- 存储函数名称：为避免与关键字冲突，存储函数名通常加上前缀 f_。
- 参数列表：可以没有参数，也可以多个参数。如果有参数，每个参数都要指定数据类型。
- 返回值类型：必须指定函数返回值的数据类型。
- 存储函数特征：指定存储函数的某些特征，可用的选项如表 7-5 所示。
- 存储函数体：存储函数体中必须至少有一条 return 语句（注意 return 和 returns 的区别）。

表 7-5 存储函数特征

选项	说明
deterministic|not deterministic	指明执行结果是否是确定的。含有日期时间函数或随机数函数时，则执行结果是不确定的。
contains sql	表示包含 sql 语句，但不包含读或写数据的语句。
no sql	表示不包含 sql 语句。
reads sql data	表示包含读数据的语句。
modifies sql data	表示包含写数据的语句。
sql security {definer | invoker}	指明谁有权限执行这个存储函数。
comment '备注内容'	指定备注。

根据存储函数体中语句数量的多少，存储函数可以分为单行语句的存储函数和多行语句的存储函数。

1. 单行语句的存储函数



附录 C 实训 7-4

例如下述语句创建一个名为 f_add 的存储函数，这是一个单行语句的存储函数。

```
Drop function if exists f_add;

Create function f_add(_a int, _b int)
	returns int no sql
	return _a + _b;
```

通过 select 查询语句使用这个存储函数的语句如下。

```
Select f_add(3, 5);   -- 使用存储函数
```

前述存储函数的返回值是一个表达式，这个表达式与 SQL 无关，所以指定存储函数特征为 no sql。返回值也可以是一个 Select 语句的查询结果，例如下述代码。

```
-- 创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”，即单元 5“5.4 聚合查询”的两张表
Use abc;

Drop function if exists f_get_total;
Create function f_get_total()
	returns decimal(9,2) reads sql data
	return
```

```
          (Select sum(col_price*col_quantity)
               from book
          );
```

其中 reads sql data 的含义是这个存储函数里有读数据的 SQL 语句。在 return 语句中的 select 语句应该用圆括号括起来，并且只能返回一个值（1 行 1 列）。

通过 select 查询语句使用这个存储函数的语句如下。

```
Select f_get_total();            -- 使用存储函数
```

2. 多行语句的存储函数

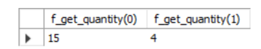
如果存储函数体有多行语句，就需要用 begin 和 end 关键字将多条语句括起来，并且用 delimiter 指定新的分隔符。

例如下述代码定义了一个存储函数，它返回指定出版社（主键）的销售数量，如果作为参数的主键值为 0，则返回所有出版社的销售数量，如图 7-6 所示。

```
Drop function if exists f_get_quantity;
-- 下一行语句参见“7.1.2 语句块和语句分隔符”
Delimiter $$
Create function f_get_quantity(_id int)
     returns int reads sql data
Begin
     If _id > 0 then
          return (Select sum(col_quantity) from book          -- 返回指定出版社的销售量
                    where id_publisher = _id
               );
     Else
          return (Select sum(col_quantity) from book          -- 返回所有出版社的销售量
               );
     End if;
End$$
Delimiter ;
```

通过 select 查询语句使用这个存储函数的语句如下。

```
Select f_get_quantity(0), f_get_quantity(1);
```



f_get_quantity(0)	f_get_quantity(1)
15	4

图 7-6 调用存储函数 f_get_quantity 的结果

7.3.2 存储函数的管理

存储函数是一个数据库对象，它与表和视图一样，保存在数据库中，因此创建存储函数后，可以在导航栏看到创建的存储函数，如图 7-7 所示。

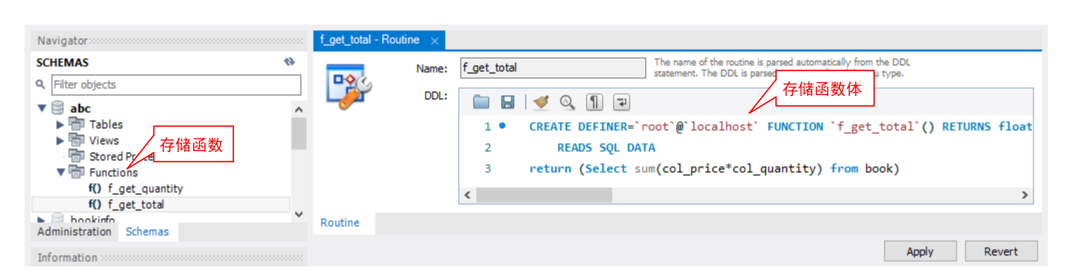
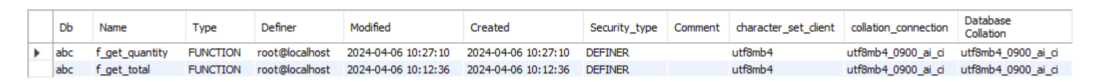


图 7-7 导航栏中的存储函数

1. 列出存储函数

列出指定数据库 abc 中的存储函数的语句如下，结果如图 7-8 所示。

```
Show function status where db = 'abc';
```



Db	Name	Type	Definer	Modified	Created	Security_type	Comment	character_set_client	collation_connection	Database Collation
abc	f_get_quantity	FUNCTION	root@localhost	2024-04-06 10:27:10	2024-04-06 10:27:10	DEFINER		utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_ci
abc	f_get_total	FUNCTION	root@localhost	2024-04-06 10:12:36	2024-04-06 10:12:36	DEFINER		utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_ci

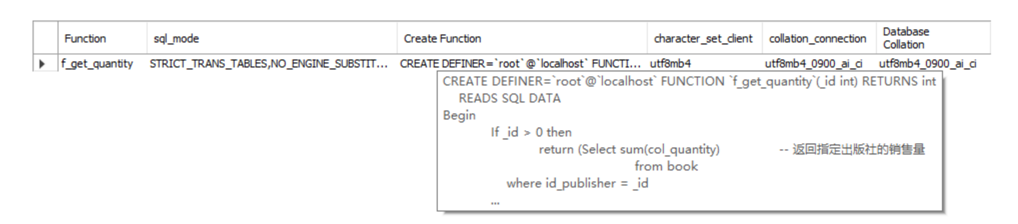
图 7-8 列出存储函数

2. 查看存储函数的定义

使用 SHOW 命令查看存储函数的定义，例如下述语句。

```
Show create function f_get_quantity;
```

由于这个存储函数是在 eshop 数据库中创建的，所以要先用 use 打开数据库。执行结果如图 7-9 所示，将光标移到 Create function 列时，会弹出一个提示框，显示存储函数的定义。



Function	sql_mode	Create Function	character_set_client	collation_connection	Database Collation
f_get_quantity	STRICT_TRANS_TABLES,NO_ENGINE_SUBSTIT...	CREATE DEFINER=\`root\`@\`localhost\` FUNCTI...	utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_ci

```
CREATE DEFINER=`root`@`localhost` FUNCTION `f_get_quantity`(_id int) RETURNS int
    READS SQL DATA
Begin
        If _id > 0 then
                return (Select sum(col_quantity)          -- 返回指定出版社的销售量
                        from book
          where id_publisher = _id
        ...
```

图 7-9 查看存储函数的定义

3. 丢弃存储函数

丢弃存储函数的语法非常简单，格式如下。

```
Drop function [if exists] 存储函数名;
```

加上 if exists 后，可以保证存储函数不存在时也不会出错。

4. 修改存储函数

修改存储函数时只能修改存储函数特征，语法格式如下。

```
Alter function 函数名 存储函数特征;
```

例如下述语句修改存储函数 f_add 的函数特征为不包含 SQL 语句，任何人可以执行。

```
Alter function f_add no sql, sql security invoker;
```

不能修改存储函数的定义，如果要修改存储函数的定义，则需要先丢弃该存储函数，然后重新创建，这也是为什么前述代码中，创建存储函数之前总是先丢弃存储函数。

7.4 存储过程

存储过程也是用得较多的一种存储程序。存储过程与存储函数有点类似，最大的区别是存储过程没有返回（return）语句，因此存储过程的使用场景与存储函数不同，存储过程不能用于 select 语句中，而是直接被调用。

7.4.1 存储过程的创建和使用

创建存储过程的语法格式与存储函数类似，格式如下。

```
Create procedure 存储过程名称(参数列表)
    [存储过程特征]
    存储过程体
```

参数说明如下。

- 存储过程名称：为避免与关键字冲突，存储过程名称名通常加上前缀 p_。
- 参数列表：可以没有参数，也可以多个参数。如果有参数，每个参数都要指定数据类型，并且还可以指定参数是输入型、输出型、或者是输入输出型的。
- 存储过程特征：指定存储过程的某些特征，与存储函数特征完全相同，参见前一节表 7-5。

- 存储过程体：与存储函数不同，存储过程体中不能有 return 语句。

根据存储过程体中语句数量的多少，存储过程可以分为单行语句的存储过程和多行语句的存储过程。

1. 单行语句的存储过程

Jitor 实训 附录 C 实训 7-5

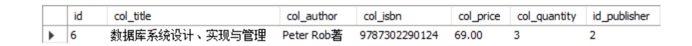
下述代码创建一个名为 p_book_by_title 的存储过程，它的作用是根据书名查询图书信息。

```
Drop procedure if exists p_book_by_title;       -- 创建表和初始化数据的代码见附录 D“项目 3a  公共代码共享”
Create procedure p_book_by_title (name varchar(50))
	Select * from book where col_title like concat('%', name, '%');
```

通过 call 关键字调用这个存储过程的语句如下。

```
Call p_book_by_title ('数据库系统');            -- 提供参数，这时的查询条件是：where col_title like '%数据库系统%'
```

执行存储过程 p_book_by_title 的结果就是执行过程体中的查询语句，结果如图 7-10 所示。对于调用者来说，不需要知道内部的细节，就可以得到查询的结果。



id	col_title	col_author	col_isbn	col_price	col_quantity	id_publisher
6	数据库系统设计、实现与管理	Peter Rob著	9787302290124	69.00	3	2

图 7-10 调用存储过程的结果

2. 多行语句的存储过程

如果存储过程体有多行语句，则需要使用语句块并指定新的分隔符。

例如下述代码根据传入的_id 值的情况，执行不同的查询。

```
Drop procedure if exists p_book_by_publisher;
-- 下一行语句参见“7.1.2  语句块和语句分隔符”
Delimiter $$
Create procedure p_book_by_publisher(_id int)
Begin
	if _id >0 then
		Select * from book
			where id_publisher = _id;
	else
		Select * from book;
	end if;
End$$
Delimiter ;
```

通过 Call 关键字用不同的参数调用这个存储过程的语句如下。

```
Call p_book_by_publisher(3);
Call p_book_by_publisher(0);
```

当传入参数为 3 时，查询出版社 id 为 3 的图书（结果有 3 行），当传入参数为 0 时，则查询所有出版社的图书（结果共有 9 行）。

7.4.2 存储过程的参数

存储过程不能像存储函数那样返回值，但是存储过程可以通过参数来返回值，这种可以返回值的参数称为输出型参数。

存储过程可以没有参数，也可以有参数。有参数时，默认为输入型参数，还可以指定参数是输出型的，或者是输入输出型的。

1. 输入型参数

见前一小节“7.4.1 存储过程的创建和使用”中的例子。

2. 输出型参数

输出型参数在变量名前加上 out 关键字。例如下述代码，这个存储过程通过参数返回总的销售金额。

```
Drop procedure if exists p_get_total;
Create procedure p_get_total(out total decimal(9,2))
        Select sum(col_price*col_quantity) into total
                from book;
```

通过 call 关键字调用这个存储过程的语句如下，需要通过一个用户变量来接收输出型参数的值。

```
Set @ammount = 0;                        -- 定义一个变量用于接收输出的数据
Call p_get_total(@ammount);
Select @ammount;
```

这个存储过程的功能与前一节的 f_get_total 存储函数在功能上是相同的，但是在使用的语法上是不同的。

3. 输入输出型参数

输入输出型参数在变量名前加上 inout 关键字。例如下述代码，参数是输入输出型的，它的输入是出版社 id 值，它的输出是该出版社的销售数量，如果_id 为 0，则是所有出版社的销售数量，输入和输出使用同一个变量_id。

```
Drop procedure if exists p_get_quantity;
-- 下一行语句参见“7.1.4 语句块和语句分隔符”
Delimiter $$
Create procedure p_get_quantity(inout _id int)
Begin
        If _id > 0 then
                Select sum(col_quantity) into _id from book         -- 再将销售数量赋值给_id（输出用）
                        where id_publisher = _id;                   -- 使用_id 作为 where 的条件（输入用）
        Else
                Select sum(col_quantity) into _id from book;        -- 将销售数量赋值给_id（输出用）
        End if;
End$$
Delimiter ;
```

通过 Call 关键字调用这个存储过程的语句如下，需要通过一个用户变量来接收输出型参数的值，同时这个用户变量也用于传入参数的值。

```
Set @id = 2;                        -- 这个变量的值用于输入
Call p_get_quantity(@id);
Select @id;                         -- 调用后同一个变量含有输出值
```

7.4.3 游标

如果想要在存储函数或存储过程中对查询结果的每一行进行处理，可以通过游标来实现。

1. 使用游标的步骤

使用游标有 4 个步骤，下面先讲解这 4 个步骤，然后再用一个例子加以说明。

1) 声明游标

游标不是数据库对象，并不保存在数据库中，因此不是用 Create 语句创建的，而是在每次使用前用 Declare 关键字来声明，这与局部变量的声明是相同的。声明游标的语法格式如下。

```
Declare  游标名  cursor for
        查询语句;
```

为避免与关键字冲突，游标名通常加上前缀 c_。

2) 打开游标

声明游标之后就可以打开游标，语法格式如下。

```
Open 游标名;
```

打开游标是把查询语句的执行结果集赋值给游标，用于下一步遍历游标。

3) 遍历游标

游标的作用是遍历（循环）访问结果集中的每一行，遍历一个打开的游标的典型代码结构如下。

```
r: loop
    fetch 游标名 into 局部变量列表;
    if _done then
        leave r;
    end if;
    -- 其他语句
end loop;
```

局部变量列表在个数和含义上必须与声明游标时的查询语句中列的个数和含义相一致。

其中，局部变量_done 是一个自定义的局部变量，用于记录捕获到的“找不到”出错信息（表示已经是最后一行），这时结束遍历。捕获出错信息的代码如下。

```
Declare continue handler for sqlstate '02000' set _done= 1;
```

上述代码即捕获到状态为'02000'的信息后（意思是找不到下一行），置局部变量_done 的值为 1，表示已经到最后一行。在循环的内部，当_done 的值为 1 时，退出循环。

4) 关闭游标

游标在使用后必须被关闭，语法格式如下。

```
Close 游标名;
```

2. 游标实例

这是一个游标实例，功能是将查询结果中各行的姓名通过 concat()函数连接起来，得到姓名列表。

```
Drop procedure if exists p_name_list;
-- 下一行语句参见“7.1.4 语句块和语句分隔符”
Delimiter %%
Create procedure p_name_list()
Begin
    declare _done int default 0;              -- 遍历结束条件，为 1 时结束遍历
    declare _list varchar(100) default '';    -- 保存结果（姓名的列表）
    declare _name varchar(20) default 0;      -- 保存每一行的姓名
    declare c_cursor cursor for               -- 声明游标
        Select name from demo;                -- 游标的查询语句，使用单元 5 的数据，见图 5-1。

    -- 捕获系统抛出的 not found 错误，如果捕获到，将 _done 置为 1 （作为结束条件）
    declare continue handler for sqlstate '02000' set _done= 1;
    open c_cursor;                                -- 打开游标
    r:  loop
        fetch c_cursor into _name;
        if _done then                             -- 退出遍历（循环）
            leave r;
        end if;
        set _list = concat(_list, ',', _name);    -- 将姓名值添加到 _list 的末尾
    end loop;
    close c_cursor;                               -- 关闭游标
    select _list;                                 -- 查询游标的计算结果
End%%
```

用下述语句调用含有游标的存储过程，结果如图 7-11 所示。

```
Call p_name_list;
```

_list
,张三,李四,王五,赵六,张七八

图 7-11 游标实例的执行结果



游标对数据库性能的影响比较大，因此应该尽量避免使用，这时可以采用其他技术来代替，例如本实例的功能可以用 group_concat()函数实现，参见“9.5.5 合并数据”。

7.4.4 存储过程的管理

管理存储过程与管理存储函数类似，只需将 function 改为 procedure 即可，不再赘述。

7.5 触发器

触发器（Trigger）是很特殊的一种存储程序，它不能被直接调用，而是当用户对表进行某些操作（插入、更改或删除行）时被自动激活。触发器类似于其他语言的事件处理机制，触发器对应的操作主要有 insert、update 和 delete 等三种。

7.5.1 触发器概述

1. 触发器的优缺点

触发器的功能十分强大，优势明显，但缺点也非常突出，应该根据项目的需求选择使用。

1) 触发器的优点

- 实现复杂约束：触发器可以实现复杂的约束。例如触发器可以引用其他表中的列，通过其他表中的数据来决定如何操作。
- 比较数据状态：触发器可以比较数据修改前后的差异，并根据这些差异采取不同的操作。

2) 触发器的缺点

- 可移植性差：不同的数据库管理系统对触发器有不同的实现，因此触发器的可移植性相当差，这是其最大的缺点。
- 占用资源：触发器占用服务器端较多的资源，对服务器造成较大的压力，有时会严重影响服务器的性能。但是要实现相同的功能，有时触发器还是比较好的选择。
- 维护困难：触发器可能造成排错困难，后期维护不方便。反过来说，由于后期维护都集中在服务器端，也方便了维护的统一管理。



鉴于触发器的缺点十分明显，许多互联网大厂明文规定禁止使用触发器，这就需要将触发器实现的功能由程序员在应用程序层面上加以实现，这也是我们要学习触发器的原因之一。

2. 触发器类型

触发器有两种类型：before 触发器和 after 触发器。这两种触发器的差别在于被激活的时机不同。

- Before 触发器：在触发它的语句之前执行，这时可以验证新数据是否满足条件，如果不满足条件，可以不执行触发语句。
- After 触发器：在触发它的语句之后执行，这时可以在触发语句执行之后完成一个或更多的操作。

3. 触发条件

触发器的触发条件有三种操作。

- Insert 触发器：在插入行的前或后时触发触发器的执行。
- Update 触发器：在更新行的前或后时触发触发器的执行。
- Delete 触发器：在删除行的前或后时触发触发器的执行。

根据触发器类型和触发条件，每张表最多有 6 个不同的触发器，如表 7-6 所示，对同一张表不能重复定义相同的触发器。

表 7-6 每张表可能拥有的触发器

触发条件	Before 触发器	After 触发器
Insert 语句	Insert 执行之前触发	Insert 执行之后触发
Update 语句	Update 执行之前触发	Update 执行之后触发
Delete 语句	Delete 执行之前触发	Delete 执行之后触发

7.5.2 创建触发器

创建触发器的语法格式如下。

```
Create trigger 触发器名 <before | after> <insert | update | delete >
        on 表名 for each row
        触发器体
```

参数说明如下。

- 触发器名：为避免与关键字冲突，触发器名通常加上前缀 t_。
- 表名：触发器是从属于表的，因此必须指定触发器从属的表名。
- 触发器类型和触发条件：指定触发器类型（before 或 after）和触发条件（insert、update 或 delete），每种组合只能有一个。
- 触发器体：触发器体中不能有 return 语句，其中的 Select 语句也不会返回给调用者。



用图形界面创建或修改触发器见下一小节的图 7-15。因为触发器是属于表的，所以位于创建表或变更表的界面中。

1. Before 触发器

Jitor 实训 | 附录 C 实训 7-6

下述代码在数据库 abc 上创建一张名为 tbl_person 的表，它的年龄列只允许输入 0-120 之间的年龄值，超过这个范围将提示出错信息。

```
Use abc;                    -- 打开数据库 abc，创建新表用于演示触发器

Drop table if exists tbl_person;      -- 丢弃表时，将同时丢弃表上的触发器
Create table tbl_person(
        id int not null primary key auto_increment,
        name varchar(20),
        age tinyint
);
```

可以用检查约束来限制年龄值为 0-120，但是本节采用触发器来实现这个功能。输入数据有两种途径：insert 和 update，因此需要写两个触发器，一个是 insert 触发器，一个是 update 触发器，因为一个 MySQL 触发器只能指定一种触发条件。这个触发器的功能是检查值的有效性，必须在插入或更新之前检查，所以两个触发器都是 before 类型的。这两个触发器的代码如下所示，两个触发器的代码基本相同。

```
Drop trigger if exists t_before_insert_person;
Drop trigger if exists t_before_update_person;
-- 下一行语句参见“7.1.2 语句块和语句分隔符” （或使用如图 7-15 所示的图形界面）
Delimiter $$
Create trigger t_before_insert_person before insert
        on tbl_person for each row
begin
        if new.age<0 or new.age>120 then
                signal sqlstate 'HY000' set message_text = "插入时，年龄范围是 0~120 之间。";
        end if;
end$$
```

```
Create trigger t_before_update_person before update
	on tbl_person for each row
begin
	if new.age<0 or new.age>120 then
		signal sqlstate 'HY000' set message_text = "更新时，年龄范围是 0~120 之间。";
	end if;
end$$
Delimiter ;
```

其中 signal sqlstate 'HY000' set message_text 用于设置出错信息，并中止激发它的语句的继续执行，在这个例子中就是分别中止插入和更新语句的继续执行。

这时不论是从图形界面，还是用 SQL 语句插入或更新 tbl_person 表，都会检查年龄值，如果检查失败，插入或更新就无法完成，并且提示出错信息，如图 7-12 所示的年龄是 122，如果年龄值超过 127，则引起的出错信息是超出 tinyint 的取值范围。

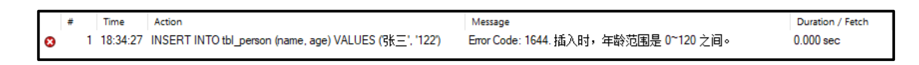


图 7-12 触发器返回的出错信息

因此，实现类似功能可以在下述三个层面上实现。

- 采用 Before 触发器实现，如上述所示。优点是位于底层，没有任何办法绕过触发器。
- 采用检查约束实现，如单元 3“3.4.6 检查约束”所示，优点也是位于底层，没有任何办法绕过检查约束。
- 由程序员编码在应用程序层面上实现，缺点是可以绕过检查，优点是减轻了数据库服务器的压力，提高了系统的整体性能，对于大数据量的项目，这个优点是非常有吸引力的。

2. After 触发器

为了演示 After 触发器，下述代码创建了两张表（tbl_person 和 tbl_log），其中第二张表用于记录操作日志。

```
Use abc;                    -- 打开数据库 abc，创建新表用于演示触发器

Drop table if exists tbl_person;                    -- 同时丢弃表上的触发器，例如上一小节的触发器
Drop table if exists tbl_log;
Create table tbl_person(
	id int not null primary key auto_increment,
	name varchar(20),
	age tinyint
);
Create table tbl_log(
	id int not null primary key auto_increment,
	log_text varchar(500),
	log_date datetime
);
```

下述代码是两个 after 触发器，可以实现对 tbl_person 表进行插入或更新操作时，在 tbl_log 表中记录操作的信息。

```
Drop trigger if exists t_after_insert_person;
Drop trigger if exists t_after_update_person;
-- 下一行语句参见“7.1.2 语句块和语句分隔符” （或使用如图 7-15 所示的图形界面）
Delimiter $$
Create trigger t_after_insert_person after insert
	on tbl_person for each row
Begin
```

```
    Insert into tbl_log values(null,
        concat('插入新行：id=', new.id, ', 姓名=', new.name, ', 年龄=', new.age), now());
End$$

Create trigger t_after_update_person after update
    on tbl_person for each row
Begin
    insert into tbl_log values(null, concat('更新行：id=', new.id,
        ', 姓名=', new.name, '(old=', old.name, '), 年龄=', new.age, '(old=', old.age, ')'), now());
End$$
Delimiter ;
```

其中的 new 和 old 在下一小节“3. 触发器中的新行和旧行”讲解。如果对 tbl_person 进行了插入或更新操作，则 tbl_log 表将记录操作的内容，如图 7-13 所示的操作的记录如图 7-14 所示。

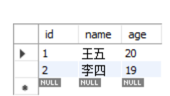


图 7-13 对 tbl_person 表的插入和更新操作

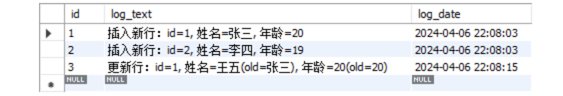


图 7-14 记录在 tbl_log 表中操作日志

从图 7-14 记录下来的数据可以看出，插入的第 1 行是“张三”，20 岁，插入的第 2 行是“李四”，19 岁，然后更新第 1 行的“张三”为“王五”，年龄没有更新。最后结果如图 7-13 所示。

对于这个例子，还可以在日志中记录删除操作，代码请读者自行补充。

3. 触发器中的新行和旧行

在触发器的触发体中，有两个特殊的对象 new 和 old。对象 new 表示将要插入的新行，对象 old 表示将要删除的旧行。在触发器中通过 new 和 old 可以方便地获取新行或旧行的列的值，并进行判断或记录。这两个对象与增删改的关系见表 7-7。

表 7-7 增删改事件中的 New 和 old 对象

事件	说明	Old	New
Insert	插入时，只有新行，没有旧行	无（不可访问）	有
Delete	删除时，只有旧行，没有新行	有	无（不可访问）
Update	更新时，是用新行替换旧行，这时新行旧行都有	有	有

对于更新操作，可以理解为将旧行替换为新行，因此这时还可以比较新行和旧行的值，根据比较的结果进行操作。

7.5.3 管理触发器

触发器是一种数据库对象，但是触发器是属于表的，因此创建触发器后，需要打开表设计界面，才能看到触发器，需要时还可以修改，如图 7-15 所示，这个界面也可以创建新的触发器。

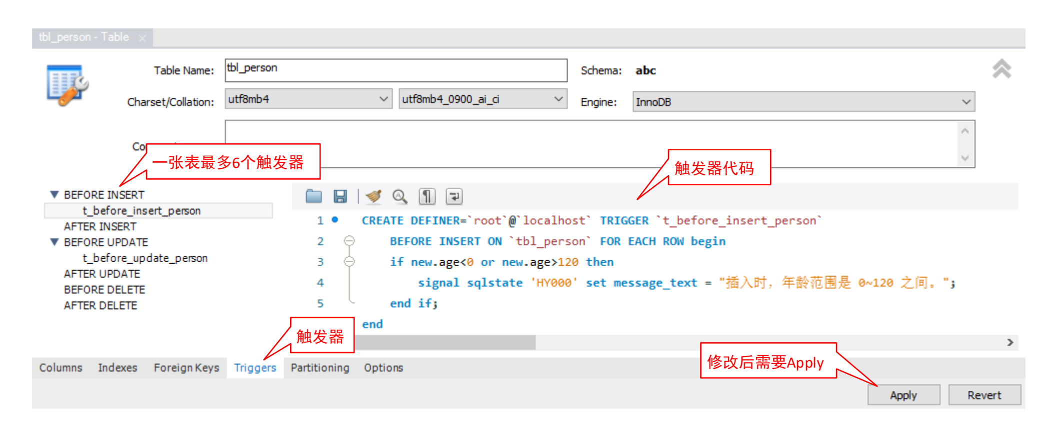


图 7-15 表设计界面上的触发器

从图 7-15 可以看到，一张表最多只能有 6 个触发器，由触发器类型（before 和 after）和触发条件（insert、update 和 delete）的组合决定。

1. 列出触发器

列出当前数据库中的触发器的语句如下。

```
Show triggers;
```

2. 查看触发器的定义

使用 Show 命令查看触发器的定义，格式如下。

```
Show create trigger 触发器名;
```

3. 丢弃触发器

丢弃触发器的语法非常简单，格式如下。

```
Drop trigger if exists 触发器名;
```

4. 修改触发器

与存储函数一样，不能修改触发器的定义，只能是先丢弃再重新创建，因此，在前述代码中，创建触发器之前总是先丢弃触发器。

与存储函数和存储过程不同，触发器没有触发器特征，更不需要修改。

7.5.4 存储函数、存储过程、触发器和事件的比较

前面 3 节分别讲解了存储函数、存储过程和触发器，下一节将讲解事件，它们都是保存 SQL 代码的数据库对象，但是有不同的特点和用途。它们的比较见表 7-8。

表 7-8 存储函数、存储过程、触发器和事件的比较

比较项	存储函数	存储过程	触发器	事件
参数	不能有 out 和 inout 参数	具有 in、out 和 inout 参数	没有任何参数	没有任何参数
返回值	必须有 return 语句	不能有 return 语句	不能有 return 语句	不能有 return 语句
调用	用 select 语句调用	用 call 语句调用	无法调用（触发时执行）	无法调用（定时执行）

7.6 事件

事件是一种在 MySQL 内部定时执行 SQL 语句的机制。一个事件有如下两方面的内容。

- 事件执行的内容：将被执行的 SQL 语句或存储过程。
- 事件执行的时间：可以指定在某个时间执行一次，也可以指定循环执行的间隔时间。

事件可以用于日常维护，例如，每日定时统计前一日的销售金额，用于制作日报表，每月月初定时统计上一个月的月报表，用于制作月报表。

7.6.1 事件的使用

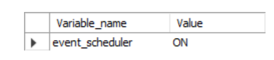
1. 启用事件



在默认情况下，事件调度是未启用的，可以通过下述语句查看当前的状态。

```
show variables like 'event_scheduler';
```

执行结果如图 7-16 所示，其值为 ON，表示已经启用。



Variable_name	Value
event_scheduler	ON

图 7-16 事件调度的状态

如果还未启用，则要在 MySQL 的配置文件的[mysqld]中加入下述设置项。

```
[mysqld]
event_scheduler=1
```

重新启动 MySQL 服务器，这时检查 event_scheduler 的值，如果是 ON，表示配置成功。

2. 创建事件

下面用一个例子来演示事件的创建，以及定时执行的效果。

先创建一张表，用于记录事件执行的效果。

```
Use abc;
Drop table if exists event_log;
Create table event_log(
	id int not null primary key auto_increment,
	col_tag varchar(20),
	col_time datetime
);
```

编写一个事件，向 event_log 表插入一行，其中 every 10 second 表示每隔 10 秒执行一次，连续执行，代码如下。

```
Drop event if exists e_log_event;
-- 下一行语句参见“7.1.2 语句块和语句分隔符”
Delimiter $$
Create event e_log_event
	on schedule every 10 second
	do
begin
	insert into event_log values(null, 'log tag', now());
end$$

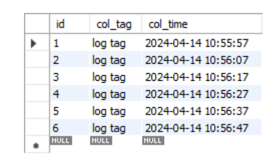
Delimiter ;
```

3. 事件执行的效果

大约 1 分多钟后查询 event_log 表的数据，可以检查事件定时执行的效果。

```
Select * from event_log;
```

图 7-17 所示是在开始事件调度后 1 分钟的查询结果，该事件共执行了 6 次，每次时间间隔是 10 秒。



id	col_tag	col_time
1	log tag	2024-04-14 10:55:57
2	log tag	2024-04-14 10:56:07
3	log tag	2024-04-14 10:56:17
4	log tag	2024-04-14 10:56:27
5	log tag	2024-04-14 10:56:37
6	log tag	2024-04-14 10:56:47
NULL	NULL	NULL

图 7-17 定时事件插入的数据

7.6.2 管理事件

事件是一种数据库对象，在数据库的“Schema Inspector”中单击 Events，可以查看所有事件的列表，如图 7-18 所示。

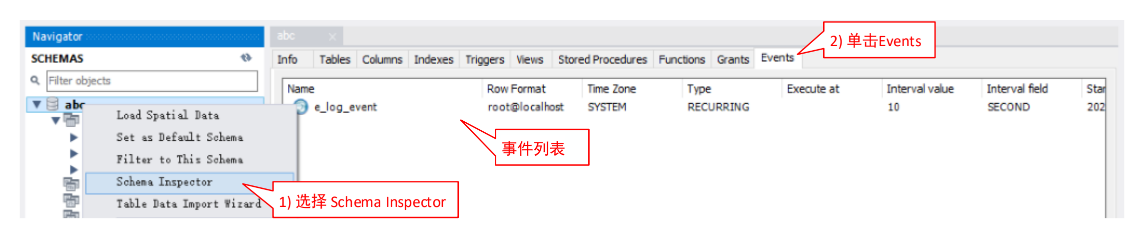


图 7-18 图形界面列出事件

1. 列出事件

列出事件的语句如下。

```
Show events;
```

2. 查看事件的定义

查看事件定义的语句如下。

```
Show create event 事件名;
```

3. 丢弃事件

丢弃事件的语句如下。

```
Drop event [if exists] 事件名;
```

4. 变更事件

如果需要变更事件的定义，可以先丢弃，然后再创建事件。

变更事件主要是失能或使能事件。可以暂时失能（禁用）事件，代码如下。

```
Alter event 事件名 disable;
```

需要时再使能事件，代码如下。

```
Alter event 事件名 enable;
```

在事件失能期间，事件是得不到执行的，当再次使能之后，才能恢复执行。

7.7 事务和锁

在单元 1 的开篇，对数据库作出如下简短的定义“数据库是存储在计算机上的有组织的、可共享的数据的集合”，可共享就是要能够在多用户环境下正常运行。

在同一时刻多个用户通过网络访问同一个数据库对象时，用户之间有可能形成冲突，从而破坏数据的一致性。事务用于在多用户环境下确保数据的一致性和完整性，锁是实现事务并发访问的一种机制。

7.7.1 事务

1. 事务的概念

事务是一个最小的不可再分的工作单元，通常它对应一个完整的业务，包含一个或多个数据库操作，如插入、更新或删除等，共同完成这个业务。

下面用一个例子来说明事务的概念，首先创建一个极其简单的银行表（bank），其中有 2 个账户 A 和 B，分别存入 2000 元和 3000 元。

```
Create table bank(       -- 说明概念的代码，无需执行
    account varchar(20) not null primary key,        -- 账户，同时作为主键用
    ammount decimal(10,2)                            -- 存款数目
);
Insert into bank values('A', 2000);                  -- 账户 A 有存款 2000 元
```

```
Insert into bank values('B', 3000);                    -- 账户 B 有存款 3000 元
Select * from bank;
```

存款总数是 5000 元。现在账户 A 想转 500 元钱给账户 B，转账操作应该如下。

```
Set @transfer = 500;                               -- 转账金额
Update bank set ammount = ammount - @transfer      -- 转账第一步，从账户 A 中减去 500 元
    where account = 'A';
    -- 特定时间点
Update bank set ammount = ammount + @transfer      -- 转账第二步，向账户 B 里加上 500 元
    where account = 'B';
Select * from bank;
```

考虑下述两种情况。

- 第一种情况：在第 1 条 Update 语句执行后，第 2 条 Update 语句还没有执行时，由于某种原因（例如停电）在这个特定的时间点出现一个致命的错误，而使第 2 条 Update 语句无法执行。这时转账没有完成，但是账户 A 的钱却被意外的扣除了。这种情况出现的机率极其微小，但并不是不可能出现，一旦出现，将给银行的信誉带来毁灭性的打击。
- 第二种情况：另一个用户（例如银行经理）想要查询银行的存款总额，如果他是在第 1 条 Update 语句执行完后的那个特定时间点进行的，那么查询到的结果是存款总额为 4500 元，而不是 5000 元。引起这个错误的原因是因为 2 个用户（转账用户和银行经理用户）的操作是在时间上极其接近的情况下执行的，而系统又没有任何的防范措施。

因此，一个完善的数据库管理系统（DBMS）必须提供一个妥善的解决方案，来正确处理上述两类事件对数据库的影响，这个解决方案就是事务，事务能够保证上述 2 条语句要么都执行，要么都不执行，并且一个事务的处理不会对其他事务造成影响。

2. 事务的特性

事务具有四个特性：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持久性（Durability）。这个四个特性也简称为 ACID（4 个单词的首字母，含义是化学中酸 acid 的意思）特性。

- 原子性：确保事务中的所有操作要么全部完成，要么全部不完成。
- 一致性：确保事务的执行结果使数据库从一个一致的状态转移到另一个一致的状态。
- 隔离性：确保多个同时执行的事务之间不会互相影响。
- 持久性：确保一旦事务提交，其结果就会被永久保存。

7.7.2 并发控制

1. 事务的并发

两个或多个事务在同一时刻（时间间隔极其短暂）访问同一个数据库对象（例如同一行）的现象称为并发。并发控制是确保在多个事务同时访问同一数据库对象时不破坏事务的 ACID 特性。

事务是并发控制的基本单位，并发控制应该保证下述两类事件发生时，DBMS 能够正常运行。

- 事务在运行过程中被强行停止（例如停电、系统崩溃）：这时，DBMS 必须保证被强行终止的事务对数据库和其他事务没有任何影响。事务通过原子性和持久性来保证在这种情况下的正常运行。
- 多个事务并发运行时，不同事务的操作交叉执行：这时，DBMS 必须保证多个事务的交叉运行，而不会产生相互影响。事务通过一致性和隔离性来保证在这种情况下的正常运行。

2. 事务控制语句

一个事务的开始、提交与回滚可以用 SQL 语句来实现，在 MySQL 中，控制事务的语句主要有三条。

1. Start transaction：显式地开启一个事务。

2. Commit：提交事务，即提交事务的所有操作。具体地说就是将事务（从 Start transaction 到 Commit 之间）所有对数据库的更新写到磁盘上的物理数据库中去，事务正常结束。
3. Rollback：回滚事务，即在事务运行的过程中发生了错误或故障，事务无法继续执行。这时将事务（从 Start transaction 到 Rollback 之间）对数据库的所有已执行的操作全部撤消，回滚到事务开始（Start transaction）时的状态。

事务的开始与结束可以由程序员使用上述控制事务语句显式控制。如果程序员没有显式地定义事务，则 DBMS 将按一定的策略自动处理事务。

MySQL 的默认事务处理策略是，将每一条 SQL 语句作为一个独立的事务，一旦执行完成，立即提交。而使用 start transaction 语句则可以定义一个事务，将多条 SQL 语句作为一个整体提交，或者在出现故障时回滚。

事务处理策略可以通过 Set autocommit 语句修改，设置 autocommit 的值为 1 时，表示自动提交，为 0 时表示不会自动提交。例如下面这段代码。

```
Set autocommit = 1;
Select * from book;
Update book set col_quantity = col_quantity+1;
Rollback;
Select * from book;
```

当设置 autocommit 为 1 时，update 语句执行后自动提交，其后的 Rollback 没有发现存在未提交的操作可以回滚，这时的结果是数量加 1。

当设置 autocommit 为 0 时，update 语句执行后并未提交，其后的 Rollback 就会将这个未提交的操作回滚，这时的结果是数量不变。



注意：对 autocommit 的设置是全局性的，非必要时不要轻易修改，默认设置为 1。

3. 事务的提交和回滚

Jitor 实训 | 附录 C 实训 7-8

下面将前一小节“7.7.1 事务”的例子加以修改，演示事务的提交和回滚。

```
-- 代码见附录 D“项目 3a 公共代码共享”
Use abc;                    -- 打开数据库 abc，创建新表用于演示事务控制

Drop table if exists bank;
Create table bank(
        account varchar(20) not null primary key,          -- 账户，主键
        ammount decimal(10,2)                              -- 存款数目
);
Insert into bank values('A', 2000);                        -- 账户 A 有存款 2000 元
Insert into bank values('B', 3000);                        -- 账户 B 有存款 3000 元

-- 存款总数是 5000 元。现在账户 A 想转 500 元钱给账户 B，这时的操作应该如下。
SET @transfer = 500;
Start transaction;
Update bank set ammount = ammount - @transfer         -- 转账第一步，从账户 A 中减去 500 元
        where account = 'A';
Update bank set ammount = ammount + @transfer         -- 转账第二步，向账户 B 里加上 500 元
        where account = 'B';
Commit;   -- 把 commit 改为 rollback ，看看有什么区别

Select * from bank;
```

每次执行上述代码都会删除表并重新创建，并初始化数据，保证每次执行时的初始数据完全相同。

分别执行上述代码两次，第一次执行时，用 commit 提交，这时的结果如图 7-19 所示。第二次执行时，将 commit 改为 rollback（回滚），这时的结果如图 7-20 所示

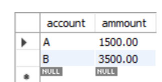


图 7-19 事务提交的结果

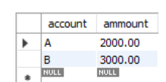


图 7-20 事务回滚的结果

一个事务从 start transaction 开始，到 commit 或 rollback 结束，其中有两条更新语句，这个事务要么都完成（commit），结果如图 7-19 所示，要么都不完成（rollback），回到事务开始时的状态，结果如图 7-20 所示，而不可能出现只完成一条语句的状态。



关于事务的实例，参见单元 9“9.4 事务的应用”。

7.7.3 锁

锁是实现事务并发访问的一种机制，其思路是，多个事务同时访问同一个数据库对象时，其中一个事务将这个数据库对象锁定，不允许其他事务访问，直到这个事务使用完毕，释放锁后，其他事务才能访问。

1. 并发问题

当多个事务并发访问时，如果没有锁，这时可能出现并发问题。依据其危害程度由高到低，列出如表 7-9 所示。

表 7-9 并发问题

并发问题	问题描述	严重程度
第一类更新丢失	两个用户更新同一数据，用户 A 已经更新了数据，用户 B 由于某种原因回滚了事务，导致用户 A 已经提交的数据被回滚的数据覆盖，这时用户 A 的更新丢失了。	极其严重
脏读	用户 A 更新了数据，用户 B 在此时读取了同一数据，用户 A 由于某种原因回滚了事务，则用户 B 所读取的数据就会是不正确的（被用户 A 弄脏）。	很严重
不可重复读	在一个事务中，对同一数据的两次查询结果数据不一致，原因是两次查询操作之间插入了另一个事务，该事务更新（update）了原有的数据。	较严重
幻影行	在一个事务中，对同一数据的两次查询结果的行数不一致，原因是两次查询操作之间插入了另一个事务，该事务插入（insert）了新行，或删除（delete）了旧行。	有点严重
第二类更新丢失	两个用户读取同一数据，并进行更新，用户 A 先更新了数据，用户 B 后更新数据，导致用户 A 已经提交的数据被用户 B 提交的数据覆盖，这时用户 A 的更新丢失了。	不严重

由于第一类更新丢失极其严重，锁机制必须防止它的出现，而第二类更新丢失最不严重，并且锁机制无法防止它的出现，必须交给应用程序来处理。因此本小节主要讨论脏读、不可重复读和幻影行。



关于第二类更新丢失，参见单元 9“9.4.2 第二类更新丢失”。

2. 锁的分类

锁的类型有许多种，可以大致分为表锁、行锁、读锁和写锁等等。

1) 按数据范围划分

按锁定的数据范围分来划分，锁可以分为表锁和行锁等。

- 表锁：锁定整张表，优点是实现简单，开销小，缺点是并发度低。
- 行锁：锁定表中一行或几行，优点是并发度高，缺点是实现复杂，开销大。

2) 按操作类型划分

按对数据操作的类型来划分，锁可以分为读锁和写锁等。

- 读锁：为读数据而加的锁，同时允许其他事务的读操作，也称为共享锁。
- 写锁：为写数据而加的锁，不允许其他事务的读和写操作，也称为排它锁。

3. 隔离级别

前述如表 7-9 所示的 5 种并发问题中，需要调整的是严重程度处于中间的脏读、不可重复读或幻读等 3 个问题，MySQL 提供了四种隔离级别，如表 7-10 所示。

表 7-10 四种隔离级别

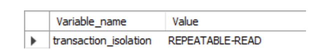
隔离级别	说明	脏读	不可重复读	幻读
Read uncommitted 未提交读	最低级别，只保证不读取物理上损坏的数据	可能有	可能有	可能有
Read committed 已提交读	语句级	避免	可能有	可能有
Repeatable read 可重复读	事务级，MySQL 的默认级别	避免	避免	可能有
Serializable 可序列化	最高级别，事务级	避免	避免	避免

MySQL 使用锁机制来实现隔离级别。需要指定 4 种隔离级别中的某一种，MySQL 会根据隔离的要求，自动选择合适的锁机制，而不需要太多的干预。

可以用下述语句查看隔离级别。

```
Show variables like '%isolation%';
```

执行结果如图 7-21 所示，显示的是 REPEATABLE-READ（可重复读）。这是 MySQL 默认的隔离级别，这个默认的隔离级别适用于大多数的需求。



Variable_name	Value
transaction_isolation	REPEATABLE-READ

图 7-21 查看隔离级别的结果

4. 锁与事务的区别

事务与锁是不同的，但又是互相关联的，它们的联系和区别如下所示。

- 事务具有 ACID 四个特性，锁用于管理事务的并发访问。
- 事务有不同的隔离级别，隔离级别是通过锁机制来实现的。
- 开启事务就是自动加锁。

【案例讲解】书店管理系统的编程



打开附录 D 的“项目 2c 书店管理项目”，看看在项目中是如何使用存储例程（存储函数或存储过程）的，然后切换到开发版，查看相关的代码。

任务 1 使用存储过程

在实战演练平台上，大量使用了 Select 语句，许多 Select 语句都可以用存储过程来代替，从而将复杂的业务逻辑隐藏在存储过程中，同时提高了代码的复用。

报表设计中的“主表数据源 Select 语句”、“从表数据源 Select 语句”，以及增删改中的“主表编辑 Select 语句”和“从表编辑 Select 语句”等，通常情况下是用 Select 语句，包括使用视图，但是也可以使

用存储过程，将可能是复杂的查询代码（例如有条件分支的代码）封装到存储过程中，然后使用“Call 存储过程(参数列表)”来调用存储过程。

例如将客户管理的主表数据源 Select 语句改写为一个存储过程，代码如下。

```
Create procedure p_customer()
    Select * from bs_customer;
```

然后将客户管理的主表数据源 Select 语句改为如下代码。

```
Call p_customer();
```

使用存储过程后，一切功能不变。

任务 3 使用事务

事务是一项极其重要的特征，本项目由于比较简单，连库存情况都没有考虑，所以不需要使用事务。在复杂一点的项目中，事务是必须考虑的，否则会出现一些意想不到的情况。实例见单元 9“9.4.1 事务的回滚”。

任务 2 使用其他存储程序

其他几种存储程序都可以在项目中使用，简单说明如下。

- 存储函数：在实战演练平台上，存储函数的使用更为直接，创建存储函数后，直接在 Select 语句中使用即可。
- 触发器：触发器与实战演练平台没有直接关系，触发器是在数据库的底层起作用的，但是也可以为项目增色不少，或简化应用程序的编程，本项目比较简单，没有合适的例子。
- 事件：事件是系统级别的存储程序，与具体的应用程序编程没有直接关系，但它可以通过定时任务，为项目提供服务。

【实战演练】图书借阅系统的编程



参考附录 D 的“项目 2d 图书借阅项目”，在读者自行开发的图书借阅项目上，在合适的地方，尝试编写 1 到多个存储例程（存储函数或存储过程）。

【单元重点】

单元小结参见本单元的【思维导图】，单元重点如下。

- 存储程序的种类、优点和缺点，语句块和语句分隔符。
- 三种变量、各种流程控制语句。
- 常用的内置函数。
- 4 种存储程序（存储函数、存储过程、触发器和事件）的作用和特点，以及它们的区别。
- 事务的概念、事务的 4 个特性（ACID 特性）。
- 并发控制的概念、事务控制语句。
- 锁在事务中的作用，隔离级别。

【课后思考】

一、选择题

1. 存储程序包括【　　　　】。

   A. 存储函数　　B. 存储过程　　C. 触发器　　D. 事件

2. 取得前一条插入语句生成的主键值，使用内置函数（　　）。

   A. count()　　B. found_rows()　　C. last_insert_id()　　D. row_count()

3. 取得前一条查询得到的行数，使用内置函数（　　）。

A. count()　　B. found_rows()　　C. last_insert_id()　　D. row_count()

4. 取得前一条 DML 语句实际影响的行数，使用内置函数（　　）。

A. count()　　B. found_rows()　　C. last_insert_id()　　D. row_count()

5. 存储函数可以在什么语句中使用【　　　　】。

A. delete　　B. insert　　C. select　　D. update

6. 存储过程的参数有（　　）。

A. 输入型　　B. 输出型　　C. 输入输出型　　D. 以上都是

7. 触发器的触发器类型有【　　　　】。

A. after　　B. before　　C. each row　　D. table

8. 触发器的触发条件有【　　　　】。

A. create　　B. delete　　C. insert　　D. update

9. 事务的特性包括【　　　　】。

A. 完整性　　B. 隔离性　　C. 并发性　　D. 持久性

10. 事务控制语句语句有【　　　　】。

A. Case　　B. Commit　　C. Rollback　　D. Start transaction

二、填空题

1. 存储例程包括_______________和_______________。
2. 调用存储过程的关键字是__________。
3. 一张表最多可以定义__________个触发器。

三、思考题

1. If 运算符和 if 条件分支语句有什么区别？Case 运算符和 case 条件分支语句有什么区别？
2. 存储函数、存储过程和触发器各自的作用是什么？它们有什么区别？
3. 什么是事务？事务有什么特点？事务可以解决哪方面的问题？

【课外拓展】

1、 阅读一篇有关事务的文章，增进对事务的作用、并发控制、事务处理的理解。

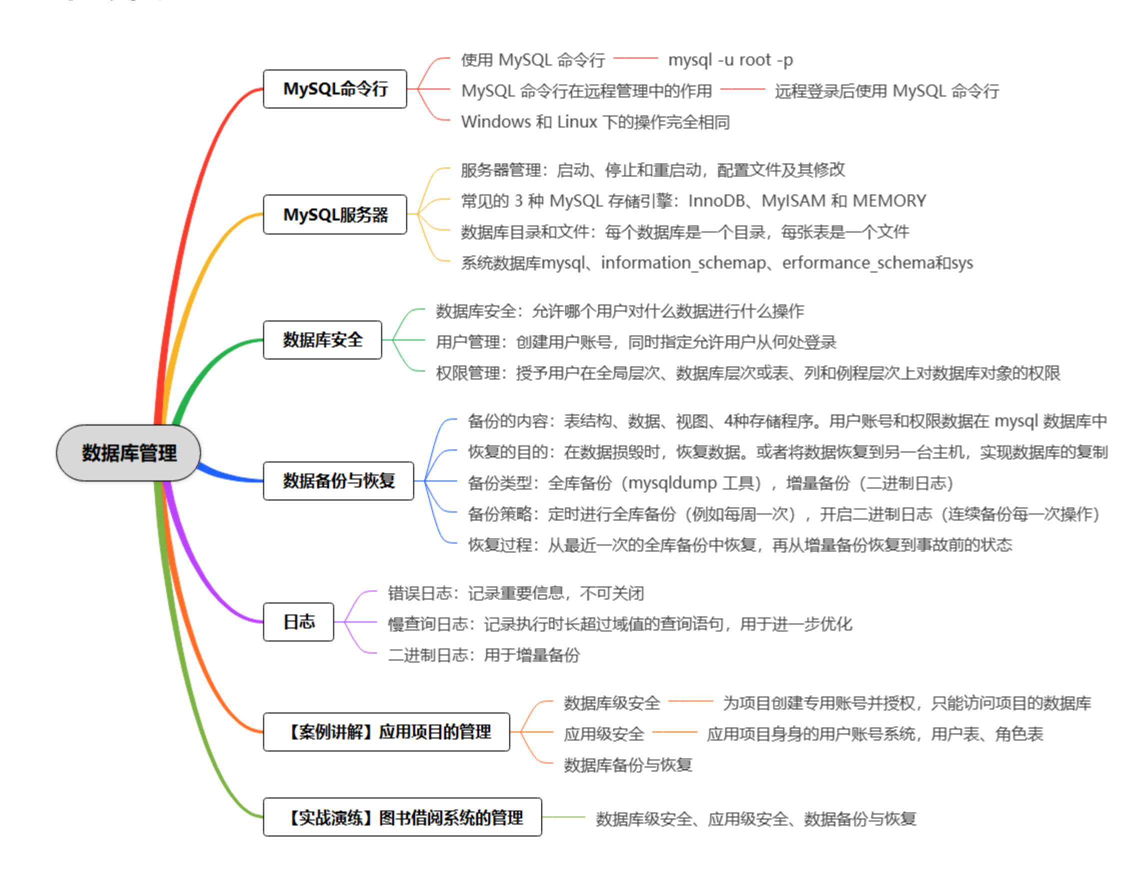
2、 阅读一篇有关存储函数、存储过程或触发器的文章，加深对它们的理解。

单元8 数据库管理

【学习目标】

知识目标	能力目标
◆ 理解 MySQL 命令行的地位和作用。	◆ 学会使用 MySQL 命令行客户端。
◆ 理解 MySQL 的远程管理。	◆ 初步学会管理用户账号，以及权限的授予。
◆ 了解 MySQL 服务器。	◆ 学会数据库备份和恢复的实施。
◆ 理解数据库安全的基本概念。	◆ 初步学会查看数据库日志。
◆ 理解数据库备份和恢复的基本概念。	**素质目标**
◆ 理解备份和恢复的策略。	◆ 建立远程管理能力、培养安全意识。
◆ 了解数据库日志。	◆ 保护用户隐私、防止数据泄露、专业道德。

【思维导图】



【情景导入】

小明学完了单元 7，真正体会到了 SQL 的强大，各种存储程序的用途各不相同，经过对存储函数、存储过程、触发器和事件进行了比较以后，小明慢慢地对数据库编程有了足够的信心，并对事务有了很好的理解，事务是多用户环境下数据一致性的有力保障。接下来是数据库管理，这是数据库运行维护的重要内容，让我们同小明一起进入这个新的领域。

【知识储备】

本单元讲解与运维有关的部分，包括 MySQL 命令行、MySQL 服务器、数据库安全、数据备份和恢复以及日志等。

8.1 MySQL 命令行

本书到目前为止，全部是使用 MySQL Workbench 作为客户端，对 MySQL 进行管理。本单元将讲解使用 MySQL 命令行对 MySQL 进行管理，本单元的操作也是在 MySQL 命令行上完成。

8.1.1 使用 MySQL 命令行

Jitor 实训 | 附录 C 实训 8-1

运行 MySQL 命令行客户端的语法如下。

```
操作系统提示符>mysql [-h  主机] [-P  端口] -u  用户名  -p[密码]
```

参数说明如下。

- 主机（host）：指定登录的主机，本地登录时不需要这个参数。
- 端口（Port）：指定 MySQL 端口号，采用默认值 3306 时不需要这个参数。注意 P 是大写的。
- 用户名（user）：指定登录的用户，通常使用 root 系统管理员用户。
- 密码（password）：登录用的密码。注意 p 是小写的，并且 p 与密码之间不能有空格，如果有空格，这个空格将被认为是密码的一部分。建议密码不要写在命令中，而是等待提示输入密码。

例如，先打开 Windows 命令行窗口（按 Win+R，然后输入 cmd），再输入如下命令。

```
mysql -u root -p
```

执行该命令后，提示输入该账号的密码，成功登录（连接）后，显示 mysql 命令行的提示符。

```
mysql>
```

这个提示符表示已经进入 MySQL 命令行，如图 8-1 所示。



图 8-1 登录 MySQL 命令行

这时可以输入任何 SQL 语句，在输入完一条 SQL 语句后，按回车键，MySQL 命令行显示执行的结果，如图 8-2 所示。

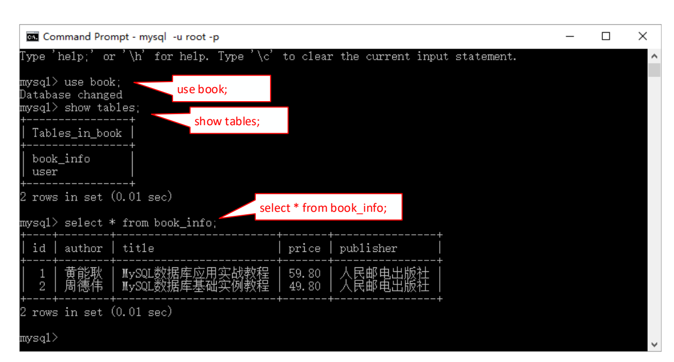


图 8-2 在 MySQL 命令行中执行 SQL 语句

语句可以分为多行输入，但语句的结束一定要加上分号，否则会提示“->”，要求继续输入语句的后续部分，如果语句已经结束，可以简单输入一个分号，告诉 MySQL 这条语句已经结束，否则会一直提示“->”，直到输入了语句的结束分号。

整个操作过程与使用 MySQL Workbench 中的 SQL 编辑窗口是类似的，不同的只是界面的形式。

MySQL 命令行的优势如下。

- 功能强大：可以完成所有 MySQL 数据库操作和日常管理工作。
- 界面简单：可以在极其简陋的条件下使用，甚至在手机上也能正常使用。
- 远程管理：这是系统管理员的首选，任何一个系统管理员都必须熟练掌握。
- 与操作系统无关：在 Windows 或 Linux 中，MySQL 命令行的使用都是相同的。



1、可以使用上下光标键调出使用过的命令，修改或不修改，按回车键再次执行。

2、退出 MySQL 命令行使用 exit 或 quit 命令，将退回到 Windows 的命令行窗口。

8.1.2 MySQL 的远程管理

实际项目的数据库全部安装在远程服务器上，即所谓的云服务器，需要远程管理，MySQL 命令行是方便的、快捷的、几乎是唯一的远程管理手段。

大多数云服务器是 Linux 服务器，同样是依靠命令行（即 Linux 终端）进行管理，例如本书作者就是通过这种方式管理 Jitor 实训教学平台的，如图 8-3 所示的是一次访问的过程，通过 PuTTY 登录 Linux，在 Linux 终端上使用 MySQL 命令行。



图 8-3 远程管理 Linux 服务器上的 MySQL 服务器

图 8-3 所示的是远程管理，对于 MySQL 命令行来说，仍然是本地登录。因为已经是以根用户的身份远程登录到 Linux，在 Linux 终端上操作，这时是在终端上进行的本地登录，无需指定主机。



Linux 根用户 root 是 Linux 上权限最高的用户，MySQL 的 root 用户是 MySQL 上权限最高的用户，但它们是两个完全不同的用户，只是名称相同而已。

8.2 MySQL 服务器

8.2.1 MySQL 服务器管理

1. 安装路径

Jitor 实训 | 附录 C 实训 8-2

MySQL 的安装路径可以通过下述命令查询得到，查询结果如图 8-4 所示。

```
Show global variables like 'basedir';
```

从图中看到安装路径是 C:\Program Files\MySQL\MySQL Server 8.0。

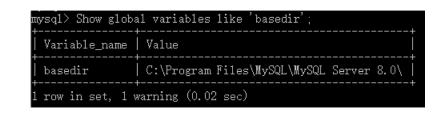


图 8-4 安装路径



图 8-5 数据文件路径

2. 数据文件路径

数据文件的保存位置可以通过下述命令查询得到，查询结果如图 8-5 所示。

```
Show global variables like 'datadir';
```

从图中看到数据文件路径是 C:\ProgramData\MySQL\MySQL Server 8.0\Data。

3. 启动、停止和重启动

MySQL 服务器的启动、停止和重启动采用命令行的方式是最简便的，如表 8-1 所示。

表 8-1 启动、停止和重启动 MySQL 服务器的命令

操作项	Windows 操作系统	Linux 操作系统
启动	net start mysql80 [注]	service mysql start
停止	net stop mysql80	service mysql stop
重启动	无，可以先停止，后启动	service mysql restart

注：服务名称 mysql80 是在单元 1 安装 MySQL 过程中的配置中指定的，参见图 1-9，默认为 mysql80。

在 Windows 操作系统下，需要以管理员身份运行“命令提示符”窗口，才有权限启动和停止 MySQL 服务器，否则提示“拒绝访问”。

在 Linux 操作系统下，需要以 Linux 根用户的身份远程登录到 Linux 上，才有权限启动和停止 MySQL 服务器。

在 Windows 操作系统下，另外一种启动、停止和重启动 MySQL 服务器的办法是通过 Windows 的“服务器管理器”管理工具，如图 8-6 所示。

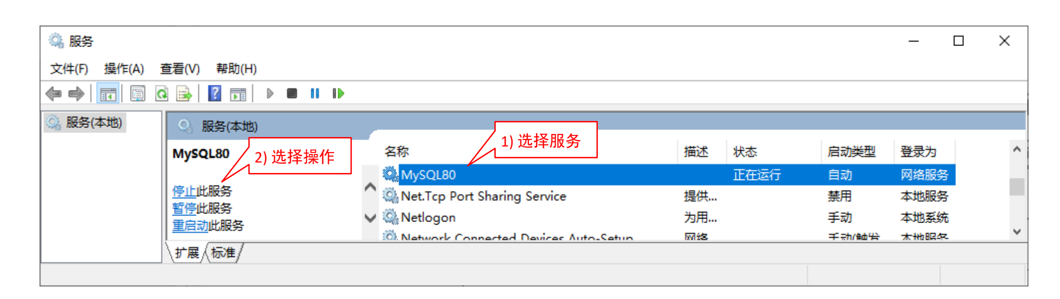


图 8-6 启动、停止和重启动 MySQL 服务器

4. 配置文件

配置文件是在 MySQL 启动时读取的，它设置了 MySQL 的全局参数以及启动参数等，其中一些参数

是在安装 MySQL 后的配置中指定的，参见单元 1 “1.2.3 安装和配置 MySQL”。

Windows 下的 MySQL 配置文件是 my.ini，位于数据文件路径下，即 C:\ProgramData\MySQL\MySQL Server 8.0 目录下。

Linux 操作系统下的配置文件是 my.cnf，位于/etc/mysql/目录下。

5. 配置文件的修改

配置文件是一个文本文件，可以用任意的文本编辑器查看和修改它，如图 8-7 所示。配置文件中以符号#起始的行是注释，每一行是一个配置项，以“键=值”的形式表示。例如下述一行。

```
port=3306
```

表示端口号配置为 3306，这是 MySQL 的默认端口号。

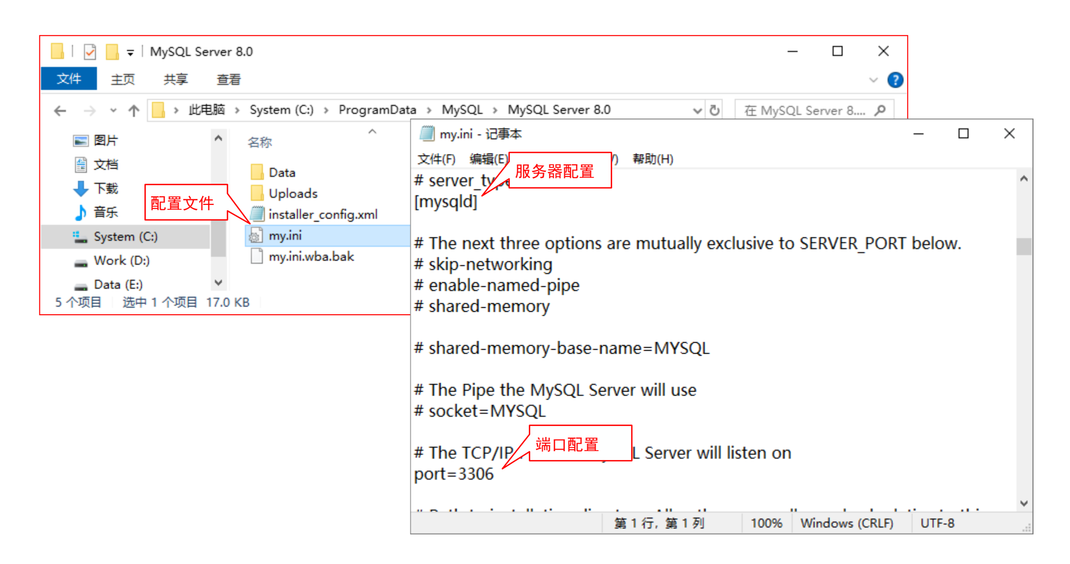


图 8-7 配置文件及配置文件的编辑

MySQL 的配置分为若干组，每一组用一个方括号括起来的标识进行标记，以下是配置文件的部分内容。

```
# CLIENT SECTION  客户端配置
# ----------------------------------------------------------------------
[client]

# pipe=

# socket=MYSQL

port=3306

[mysql]
no-beep

# default-character-set=

# SERVER SECTION  服务器端配置
# ----------------------------------------------------------------------
# server_type=3
[mysqld]

# The TCP/IP Port the MySQL Server will listen on
port=3306
```

```
# Path to installation directory. All paths are usually resolved relative to this.
# basedir="C:/Program Files/MySQL/MySQL Server 8.0/"

# Path to the database root
datadir=C:/ProgramData/MySQL/MySQL Server 8.0/Data

# The default character set that will be used when a new schema or table is
# created and no character set is defined
# character-set-server=
```

因此修改 MySQL 服务器的配置，应该在配置文件的[mysqld]内进行修改，找到需要配置的项，更改配置的值。如果配置项已被注释，则要取消行首注释符号#，再作必要的修改即可。

配置文件修改后，保存文件，然后还要重启 MySQL 服务器，修改后的配置才能生效。

8.2.2 MySQL 存储引擎

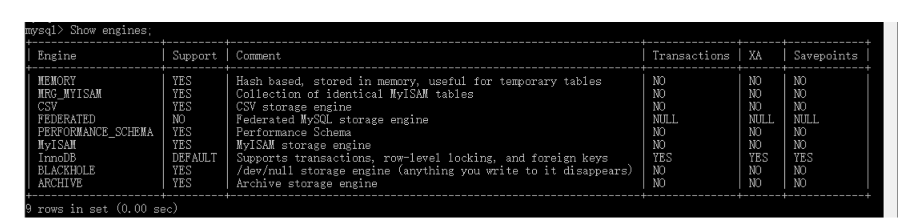
存储引擎是指表在数据库文件中的存储方式，不同的存储方式会影响存储策略、索引技巧、锁机制等底层技术。有的引擎性能高，但功能弱，有的引擎性能低，但功能强。因此，选择合适的存储引擎，可以得到所需要的性能和功能，以达到一种平衡。

1. 查看 MySQL 支持的存储引擎

使用下述命令查询存储引擎的列表。

```
Show engines;
```

查询结果如图 8-8 所示。



```
mysql> Show engines;
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| Engine             | Support | Comment                                                        | Transactions | XA   | Savepoints |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO   | NO         |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          | NO           | NO   | NO         |
| CSV                | YES     | CSV storage engine                                             | NO           | NO   | NO         |
| FEDERATED          | NO      | Federated MySQL storage engine                                 | NULL         | NULL | NULL       |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | NO           | NO   | NO         |
| MyISAM             | YES     | MyISAM storage engine                                          | NO           | NO   | NO         |
| InnoDB             | DEFAULT | Supports transactions, row-level locking, and foreign keys     | YES          | YES  | YES        |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | NO           | NO   | NO         |
| ARCHIVE            | YES     | Archive storage engine                                         | NO           | NO   | NO         |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
9 rows in set (0.00 sec)
```

图 8-8 MySQL 支持的存储引擎

2. MySQL 存储引擎介绍

图 8-8 列出了所有的存储引擎，下面对常用的几种进行介绍。

1) InnoDB

这是最常用的一种引擎，也是 MySQL 8.0 的默认引擎，本书所有的数据库都采用了 InnoDB 引擎。它的特点如下。

- 支持外键约束：其他的引擎都不支持外键约束。
- 支持事务：只有 InnoDB 引擎支持事务，具有提交或回滚等事务处理能力。
- 具有灾难恢复的能力。
- 适用于存在大量并发的写操作的数据库。
- 不支持全文索引。

2) MyISAM

这是一种旧的引擎，也是 MySQL 5.1 及之前版本的默认引擎。MyISAM 引擎比较简单，不支持外键约束和事务，不适用于存在大量写操作的数据库。它的特点如下。

- 不支持外键约束和事务。
- 不具有灾难恢复的能力。

- 适用于存在大量读操作的数据库，查询性能较好。
- 支持全文索引。

3) MEMORY

这是一种基于内存的引擎，所有数据都在内存中，避免了对硬盘读写操作，因此效率特别高。它的特点如下。

- 效率高，查询速度极快。
- 受内存容量的限制，只适用于数据较少、需要频繁访问的数据库。

3. 设置存储引擎

MySQL 8 的默认存储引擎是 InnoDB，需要时可以使用其他存储引擎。

1) 创建表时指定存储引擎

创建表时可以指定存储引擎，例如下述代码指定新创建表的引擎为 MyISAM。

```
Create table 表名 (
    列定义
) engine=MyISAM;
```

2) 修改表的存储引擎

也可以修改表的存储引擎，例如下述代码修改表的引擎为 MyISAM。

```
alter table 表名 engine=MyISAM;
```

如果表的行数非常大，则修改表的存储引擎会非常耗时，建议先复制空表、修改存储引擎，再复制数据。

8.2.3 MySQL 数据库的组成

1. 数据库目录和文件

在 MySQL 服务器上，使用 InnoDB 引擎时，每个数据库是一个目录，位于数据文件路径中，每张表对应一个文件，如图 8-9 所示。

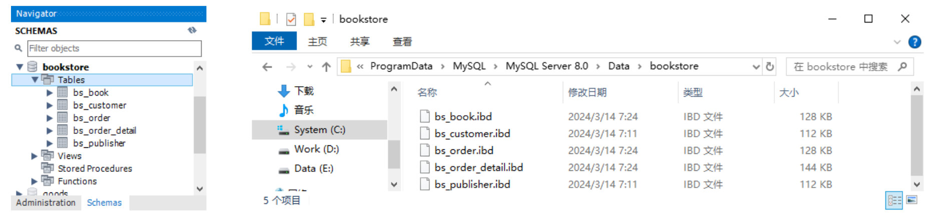


图 8-9 数据库中的每张表对应一个文件（以 bookstore 数据库为例）

在不同的操作系统下，这些文件是相同的。但是，Linux 操作系统的目录名和文件名是区分大小写的，这导致 Linux 操作系统下的数据库名、表名也是区分大小写的。

2. 系统数据库

MySQL 的数据库分为系统数据库和用户创建的数据库，系统数据库有 mysql、information_schema、performance_schema 和 sys 等。它们是在安装后，首次配置时自动创建的。它们保存了 MySQL 服务器的关键数据，具有非常重要的作用，所以千万不能删除它们。

8.3 数据库安全

数据库安全是指允许合法用户的合法访问，而拒绝非法用户的访问或合法用户的非法访问，从而保护数据不被泄露、不被篡改、不被破坏。简单的说，就是允许哪个用户对什么数据进行什么操作。

MySQL 数据库安全涉及到下述四个方面。

- 是哪个用户：MySQL 可以管理用户账号，只允许合法的用户登录。
- 从什么途径：MySQL 对用户的登录途径作出限制，例如只允许用户从本地登录。
- 对什么资源：指定可以访问的资源，资源是指数据库对象，即数据库、表、存储例程（存储函数及存储过程）、视图等。例如可以访问哪一个数据库、哪张表、哪个存储函数或存储过程。
- 做什么操作：指定具体的操作，对于表和视图，应指定是否有权查询，还是更新、插入或删除数据，对于存储函数和存储过程，则是指定是否有权运行。

8.3.1 数据库安全概述

1. 权限分类

对不同的数据库对象有不同的权限，例如对某张表是否有权查询、插入数据、更新数据和删除数据，一些常用的权限如表 8-2 所示。

表 8-2 数据库、表和存储例程（存储函数及存储过程）的权限

数据库对象	权限设置
数据库	Create, Alter, Drop, Grant, References 等
表	Select, Insert, Update, Delete, Create, Alter, Drop, Grant, References, Index 等
存储例程	Execute, Create routine, Alter Routine 等

2. 授权过程

权限管理的原则是：拥有授权即可为，没有授权不可为。就是说，任何一项权限都必须有明确的授权，否则就是非法的。

授权的过程包括两个阶段：身份验证和权限核实。

1) 身份验证

身份验证就是用户登录（连接）的验证过程，用户从 MySQL 客户端或其他用户界面（例如应用程序的连接代码）提交账号和密码信息登录（连接）到 MySQL 服务器。

2) 权限核实

MySQL 是一个多用户的数据库管理系统，允许不同的用户登录，不同的用户具有不同的权限。根用户（root）是系统管理员账号，拥有最高权限，而其他用户通常具有较低的权限。

因此，当一个用户登录后，就需要在 mysql 数据库（MySQL 的内置数据库）中与安全有关的表中查找，核实所获得的权限，这个过程分为三个层次。

- 全局层次：在用户表（user）中查询权限，如果获得权限，即认可这个授权，如果没有获得权限，则要进行数据库层次的授权。
- 数据库层次：在数据库表（db）中查询针对该数据库的权限，如果获得权限，即认可这个授权，如果没有获得权限，则要进行表、列和存储例程层次的授权。
- 表、列和例程层次：在 tables_priv、columns_priv、procs_priv 表中查询针对指定数据库的表、列、存储例程（存储函数、存储过程）的权限，如果获得权限，即认可这个授权
- 如果在上述三个层次都没有获得权限，则拒绝授权。

8.3.2 用户管理

1. 创建用户账号



创建用户账号的语法格式如下。

```
Create user 账号名@主机 identified by '密码';
```

参数说明如下。

- 账号名：用户的账号名，完整的账号名称是“账号名@主机”，在管理账号和授权时需要使用完整的账号名称。
- 主机：允许从什么主机登录，如果主机是 localhost，则只允许该用户从本机登录，如果主机是%，则允许从任意主机登录。还可以指定主机为某个 IP 地址，这样就只能从这个 IP 地址登录。
- 密码：用户账号的密码，要求至少 4 个字符长。

例如下述代码创建一个名为 zhangsan 的用户账号，完整的账号名称是“zhangsan@localhost”，因此该账号只能从本机登录。

```
Create user zhangsan@localhost identified by '123456';
```

在 MySQL 命令行上创建账号如图 8-10 所示。

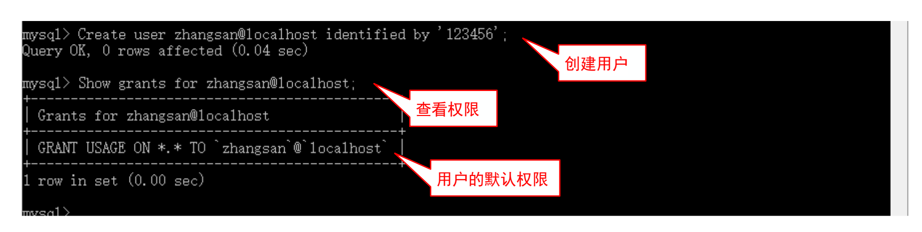


图 8-10 在 MySQL 命令行上创建账号以及用户的默认权限

新用户默认拥有 USAGE 权限，这是一个连接（登陆）权限，拥有这个权限的用户只能连接（登录）到数据库服务器，除此之外，不能执行任何操作。所有用户自动拥有 USAGE 权限，该权限不能被撤回。

这时当前用户应该是 root，或者需要拥有创建用户的权限，否则会提示权限不够。



出于安全考虑，建议只允许本地登录。进行远程管理时，也是先远程登录到安装了 MySQL 的服务器上（通常是 Linux），再以本地登录的方式使用 MySQL 命令行的，参见图 8-3。

2. 列出用户账号

列出所有用户账号需要访问 mysql 数据库的 user 表，语句如下，如图 8-11 所示。

```
Select user, host from mysql.user;
```

其中表名 user 前加上数据库名称 mysql（用小数点分隔），用于指定表所属的数据库，从而避免用一条单独的 MySQL 命令“use 数据库名;”来切换数据库。

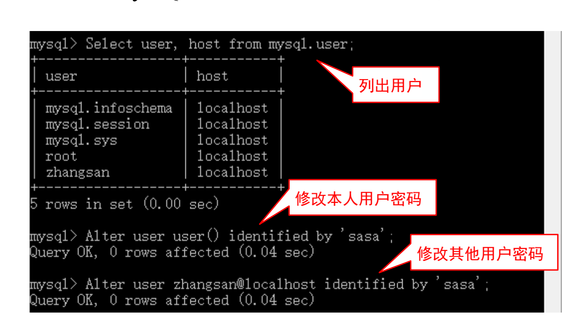


图 8-11 列出用户和修改密码

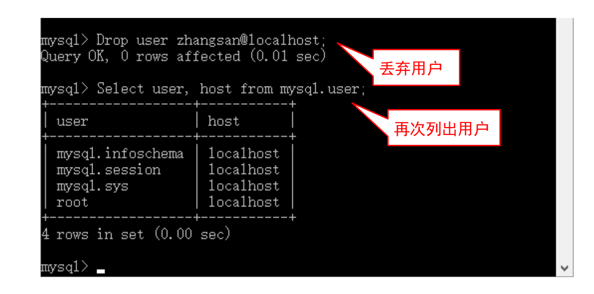


图 8-12 丢弃用户和再次列出用户

3. 修改用户密码

1) 修改当前用户的密码

例如，修改当前用户的密码的代码如下，如图 8-11 所示。

```
Alter user user() identified by 'sasa';
```

其中 user()表示当前用户，单引号括起来的部分是新密码。当前用户的密码不需要特别的权限。

2) 修改其他用户的密码

例如，修改其他用户（zhangsan@localhost）的密码的代码如下，如图 8-11 所示。

```
Alter user zhangsan@localhost identified by 'sasa';
```

这时当前用户需要拥有修改密码的权限，否则会提示权限不够。

4. 丢弃用户账号

例如，丢弃刚才创建的用户的代码如下，如图 8-12 所示，再次列出用户证实用户已被丢弃。

```
Drop user zhangsan@localhost;
```



创建用户、修改其他用户的密码或丢弃用户等与安全有关的操作都需要较高的权限，要以根用户（root）身份登录后才能操作。

5. 图形界面管理用户

可以在 Workbench 的图形界面管理用户，如图 8-13 所示。

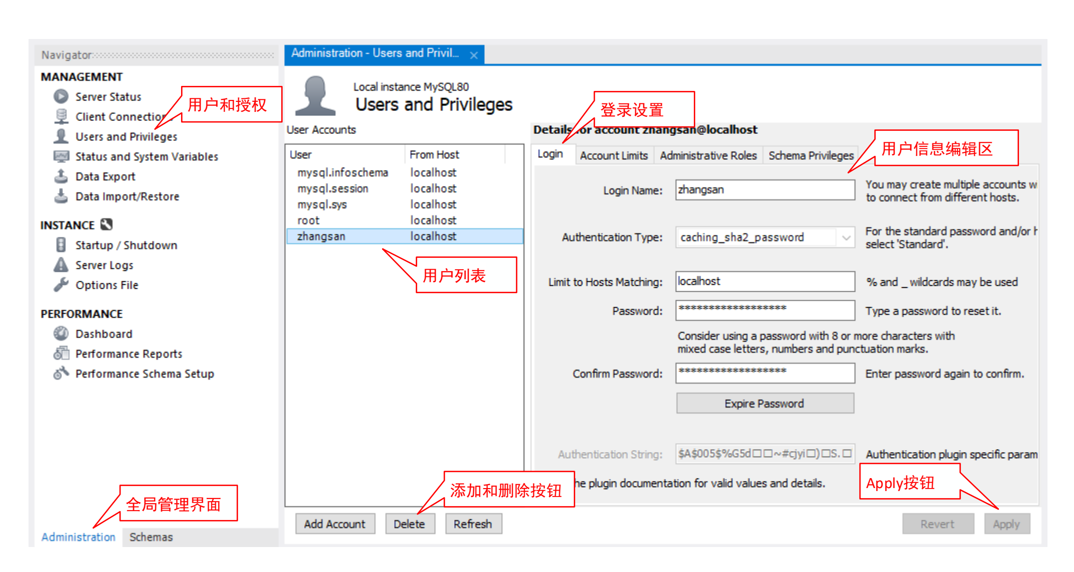


图 8-13 用图形界面管理用户

6. 角色管理

角色是一组权限的集合，一个用户拥有了某个角色的身份，便拥有了这个角色的权限。

角色与用户是多对多的联系，一个用户可以拥有多个角色，一个角色也可以授予多个用户，从而方便了用户权限的管理。

MySQL8 内置了 11 个管理角色，如图 8-14 所示。系统管理员拥有这些角色的身份，所以拥有最高的权限。如果授予某个用户拥有这些角色的身份，那么这个用户也拥有最高的权限。

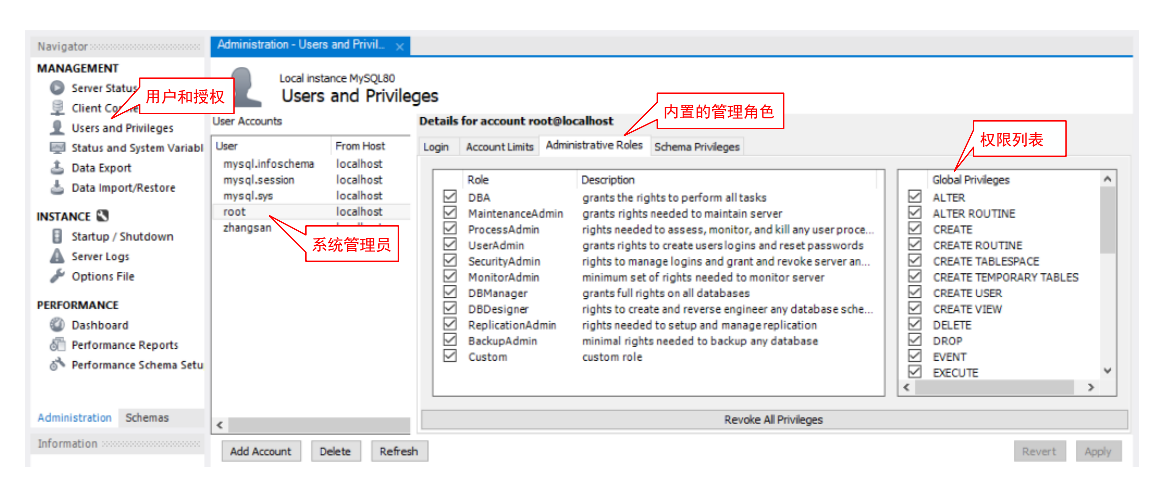


图 8-14 内置的管理角色

8.3.3 权限管理

1. 授予权限

授予权限的语法格式如下。

```
Grant 权限列表 on 数据库名.表名 to 用户@主机;
```

参数说明如下。

- 权限列表：权限是一些与 SQL 命令相关的操作，例如 Select、Insert、Delete 和 Create 等，多个权限之间用逗号分隔。一些常用的权限如表 8-2 所示。
- 数据库名.表名：允许用户操作的数据库对象，参见表 8-3。
- 用户@主机：被授予权限的用户的完整的用户账号名称。

例如授予用户 zhangsan@localhost 访问 abc 数据库的 demo 表查询、删除权限，代码如下。

```
Grant select, delete on abc.demo to zhangsan@localhost;
```

如果要授予所有权限，可以用 all privileges 表示所有的权限。

2. 权限的层次

根据授予权限时“on 数据库名.表名”的写法，权限可以分为如表 8-3 所示的三个层次，这些授权分别保存在 mysql 数据库的 user 表、db 表、tables_priv 表等。

表 8-3 权限的三个层次

层次	授权语法	说明	保存权限的表
全局层次	on *.*	适用于所有数据库	mysql.user
数据库层次	on 数据库名.*	适用于指定的整个数据库	mysql.db
表、列和例程层次	on 数据库名.表名	适用于指定的表	mysql.tables_priv
	…(列名) on 数据库名.表名	适用于指定表的指定列	mysql.columns_pri
	on 数据库名.例程名	适用于指定的例程	mysql.procs_priv

核实权限时，将根据表 8-3 所示的三个层次从上向下进行核实，一旦获得授权，表明获得相应的授权，如果在三个层次都没有获得权限，则拒绝授权。

3. 查看权限

例如查看用户 zhangsan@localhost 所拥有的权限。

```
Show grants for zhangsan@localhost;
```

执行结果如图 8-15 所示，其中 USAGE 是每个用户都有的默认权限，不能撤回的。

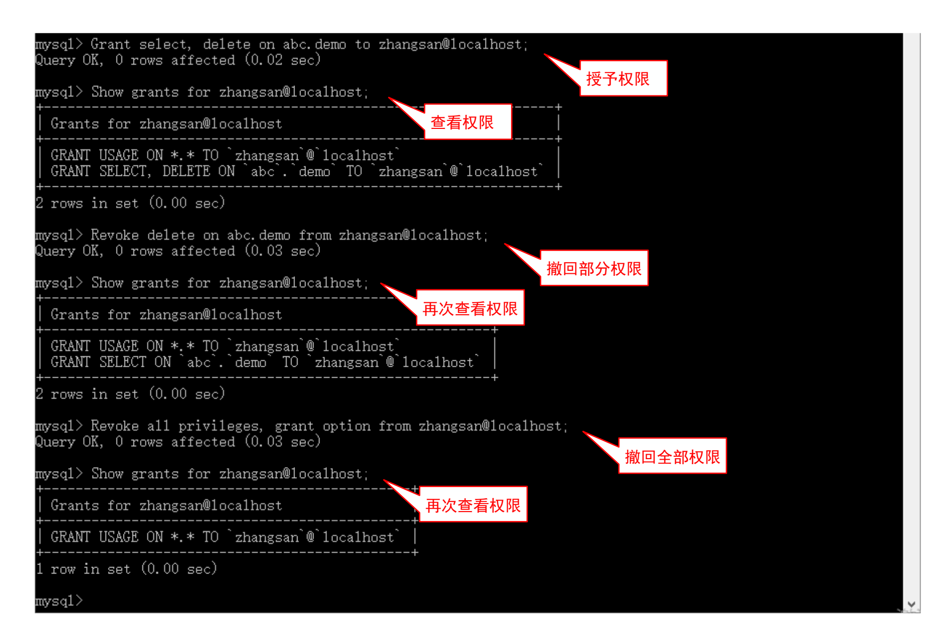


图 8-15 授予权限、查看权限与撤回权限的操作过程

4. 撤回权限

撤回权限可以是撤回部分权限。例如撤回用户 zhangsan@localhost 对 abc 数据库的 demo 表删除权限。

```
Revoke delete on abc.demo from zhangsan@localhost;
```

这时查看 zhangsan@localhost 的权限，就只剩下 select 权限，如图 8-15 所示。

撤回权限也可以是撤回全部权限。例如撤回用户 zhangsan@localhost 的全部权限。

```
Revoke all privileges, grant option from zhangsan@localhost;
```

这时已没有任何权限，仅有 USAGE 权限（即保留登录权限），如图 8-15 所示。

5. 图形界面管理授权

可以在 Workbench 的图形界面管理授权，如图 8-16 所示。

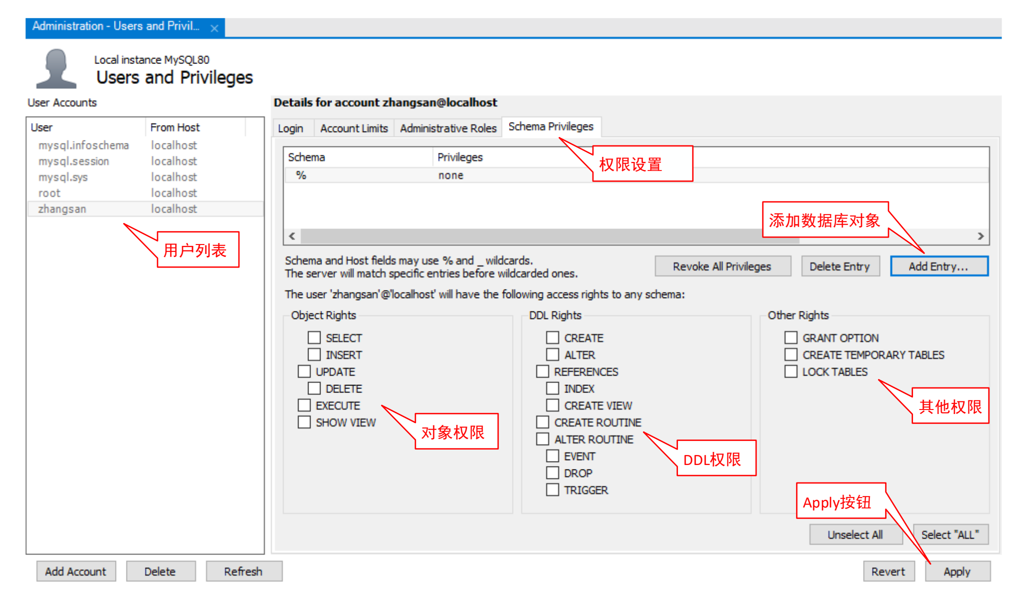


图 8-16 用图形界面管理授权

8.3.4 应用项目的安全

应用项目的安全包括数据库级的安全和应用项目级的安全两个方面，详见“【案例讲解】应用项目的

管理”中的讲解。

8.4 数据备份和恢复

8.4.1 数据库备份概述

在数据库运行的过程中，可能会由于人为的、意外的或者是不可抗的因素，造成数据的损坏或丢失，例如一场火灾把一家银行的数据库机房烧毁了，而引起数百万人的存款和贷款数据的丢失，这种情况是绝对不允许出现的。造成数据损坏或丢失的可能原因有如下几种。

- 自然或人为灾害：例如地震、水灾、台风、火灾和战争等自然或人为灾害。
- 系统硬件故障：例如磁盘损坏，雷击或其他硬件故障导致的数据损坏。
- 系统软件故障：例如操作系统或应用软件的故障引起的数据损坏。
- 计算机病毒：有些计算机病毒会故意损坏数据，包括数据库的数据。
- 黑客攻击：黑客的恶意攻击，可能造成数据丢失或被篡改。
- 人为误操作：人为误操作是经常发生，例如误删除了表或修改了数据。

防止数据损坏或丢失的措施有数据备份、双机热备、异地存储、远程集群等多种方案，本书仅讲解最基本的数据备份。

1. 备份的内容

通常情况下，备份的内容是 MySQL 服务器上的一个数据库或全部数据库的下述数据库对象，如图 8-17 所示。触发器是属于表的，事件是属于整个数据库系统的，所以没有在图中出现。

- 表（Tables）：包括数据结构、数据等所有数据，以及索引的定义等。
- 视图（Views）：视图的定义，视图本身并没有数据。
- 存储过程（Stored Procedures）：存储过程的定义，即 SQL 编写的代码。
- 存储函数（Functions）：存储函数的定义，即 SQL 编写的代码。
- 触发器（Triggers）：触发器的定义，即 SQL 编写的代码。
- 事件（Events）：事件的定义，即 SQL 编写的代码。

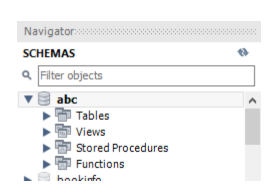


图 8-17 数据库对象

当备份一个数据库时，不包括用户账号、权限控制等数据，因为这些数据保存在 mysql 数据库中，需要备份 mysql 数据库或备份全部数据库时才能备份用户账号和权限控制等数据。

另外，可能还需要再备份 MySQL 服务器的配置文件，以便重新安装 MySQL 时，可以重新构建一个完全相同的 MySQL 服务器。

2. 备份类型

1) 逻辑备份与物理备份

逻辑备份是指以文本的方式（SQL 语句），把数据库的数据结构和数据备份为 SQL 脚本文件，这个文件的内容是由 Create 和 Insert 等语句组成的，从这些语句可以完整的恢复数据库的数据结构、索引、存储函数、存储过程、触发器，及所有数据。

物理备份是指对数据库文件进行的备份，它仅仅是复制文件，因此备份速度快。但是只能在 MySQL

服务器停止服务时才能进行，并且只能移植到相同版本的 MySQL 服务器上。

2) 热备份与冷备份

热备份是指数据库服务仍然在运行时所做的备份，在备份的过程中，服务器仍然能够提供正常的读写操作，提供正常的服务。

冷备份是指数据库服务必须暂停下来，并将所有未写入数据库的数据写到数据库文件中，然后进行的备份。

8.4.2 数据库备份

本书仅讲解在运维中最常使用的 mysqldump 命令，它是逻辑备份也是热备份。

Jitor 实训	附录 C 实训 8-4

1. 数据库备份

数据库备份使用 mysqldump 命令，其语法格式如下。

操作系统提示符>***mysqldump -u 用户名 –p[密码] [选项] 数据库名 > 脚本文件名***

这条命令必须在 Windows 或 Linux 操作系统中进行。在 Windows 下，则是打开命令提示符窗口，输入上述命令。

参数说明如下。

- 用户名：有权限进行数据库备份的用户账号，通常是系统管理员（root）。
- 密码：登录用的密码，-p 和密码之间不能有空格，建议密码不要写在命令中，而是等待提示输入密码。
- 选项：常用的选项有-d（仅备份表结构）、-t（仅备份数据）和-R（备份存储例程），不加任何选项时会备份除存储例程、事件之外的所有数据库对象。
- 数据库名：要备份的数据库的名称。
- 脚本文件名：用于保存备份的本地文件。要用一个文件重定向元字符“>”（大于号）将备份的输出重定向到指定的文件。

例如下述命令备份数据库 abc 到 abc.sql 文件中(其中 sasa 是 root 的密码)，操作过程如图 8-18 所示。

```
mysqldump -u root -psasa abc > abc.sql
```



图 8-18 在 Windows 命令提示符窗口中进行备份操作

备份成功后，打开 abc.sql 文件，其内容如下。

```
-- MySQL dump 10.13    Distrib 8.0.19, for Win64 (x86_64)
--
-- Host: localhost        Database: abc
-- ------------------------------------------------------
-- Server version 8.0.19

备份文件的内容很长，完整内容见附录 D“项目 3a 公共代码共享”的“单元 8 数据库编程”，
```

备份文件中的内容是单元 5“5.1 单表查询”使用的数据，从备份文件的内容来看，备份文件中包含了所有创建表语句（CREATE TABLE），以及插入数据语句（INSERT INTO），通过这些语句，可以复原一个完整的数据库。

2. 用图形界面进行备份

可以在 Workbench 的图形界面进行备份（导出），如图 8-21 所示，导出完成后显示相关信息。

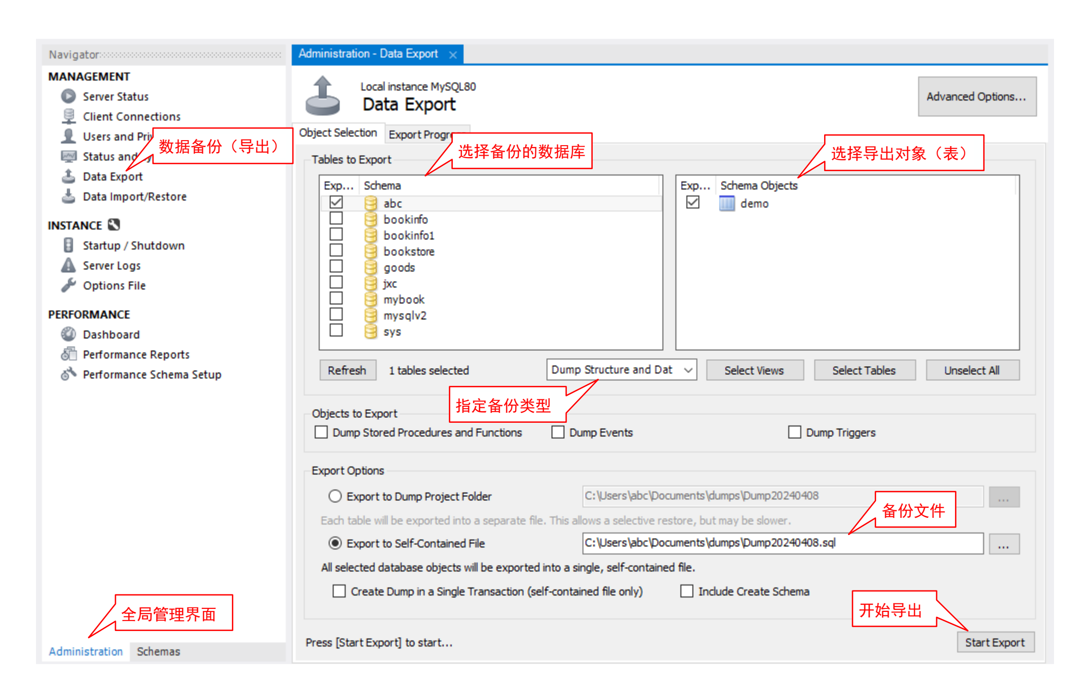


图 8-19 用图形界面进行备份（导出）

8.4.3 数据库恢复

1. 数据库恢复

数据库恢复使用 mysql 命令（该命令就是 MySQL 命令行），其语法格式如下。

操作系统提示符>***mysql -u 用户名 –p[密码] 数据库名 < 脚本文件名***

这条命令必须在 Windows 或 Linux 操作系统中进行。在 Windows 下，则是打开命令提示符窗口，输入上述命令。

参数说明如下。

- 用户名：有权限进行数据库备份的用户账号，通常是系统管理员（root）。
- 密码：登录用的密码，-p 和密码之间不能有空格，建议密码不要写在命令中，而是等待提示输入密码。
- 数据库名：要备份的数据库的名称。
- 脚本文件名：用于保存备份的本地文件。要用一个文件重定向元字符“<”（小于号）将“脚本文件”的内容作为 mysql 命令的输入。

例如下述命令将备份文件 abc.sql 文件中的数据恢复到数据库 abc1 中（sasa 是 root 的密码，需要先创建数据库 abc1），操作过程如图 8-20 所示（图示的过程相当于是复制了数据库）。

```
mysql -u root -psasa abc1 < abc.sql
```

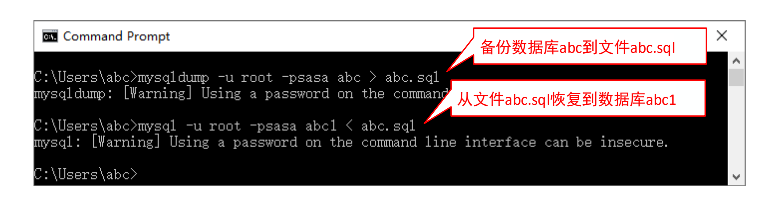


图 8-20 在 Windows 命令提示符窗口中进行备份和恢复操作

在恢复数据库之前，如果数据库不存在，应该先创建数据库，如果数据库已存在，则恢复时会通过丢弃表，重新创建表，插入数据的方式来自动恢复所有数据。但是如果存在备份文件中不存在的表，则这部分内容保持不变。

因此用备份文件完全覆盖一个现有的数据库的操作如下。

- 丢弃现有数据库：该数据库可能是已损坏的数据库。
- 重新创建新的数据库：这是新的空的数据库。
- 恢复数据库：从备份文件中恢复数据到新创建的数据库。

2. 用图形界面进行恢复

可以在 Workbench 的图形界面进行恢复（导入），如图 8-21 所示，导入完成后显示相关信息。

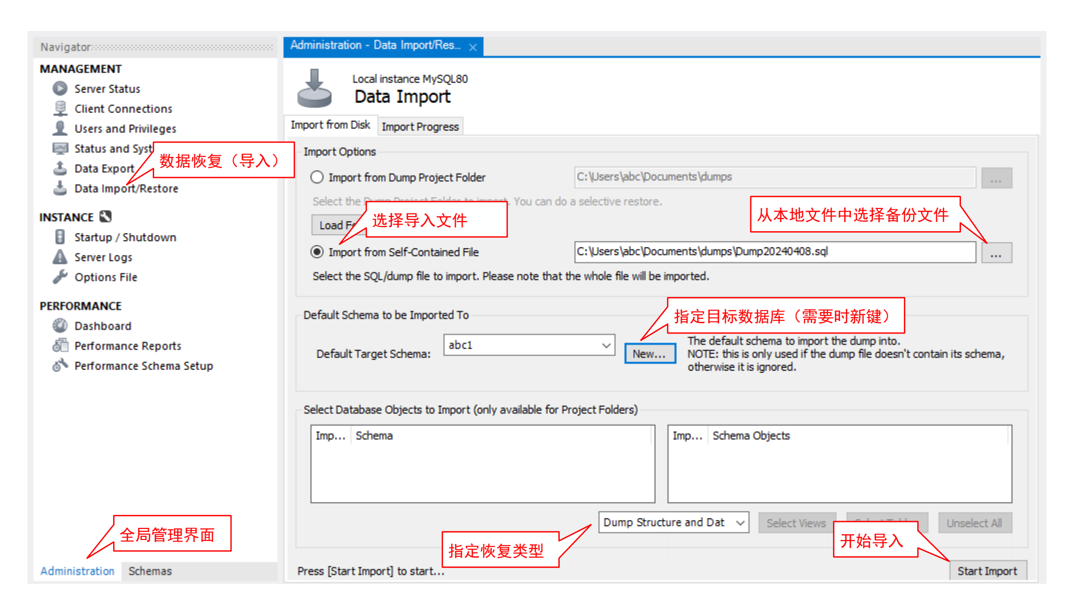


图 8-21 用图形界面进行恢复（导入）

8.4.4 备份策略和恢复策略

前述数据库备份虽然是热备份，不需停止 MySQL 服务就能进行，但它是一种静态的备份策略，是在指定的时间对整个数据库进行备份，因此称为全库备份。

数据库的数据是动态变化的，每时每刻都可能有数据的插入、更新或删除，而这些数据的变化是十分重要的，不允许有任何的闪失，定时的、对整个数据库的备份显然无法做到对每一个数据变化的备份。

1. 备份策略概述

因此需要建立一种完善的备份策略来实现对每一个数据变化都能得到完美的备份。这个策略是由两种不同形式的备份来共同完成的，如表 8-4 所示。

表 8-4 备份策略与两种备份形式

备份形式	说明	备份时间	采用的技术
全库备份	对整个数据库进行的备份，作为数据库恢复的基础	定时备份，如每周或每天一次	mysqldump 命令
增量备份	连续备份每一次修改，不间断地备份数据的变化	即时备份，持续进行	二进制日志

全库备份是在操作系统下采用 mysqldump 命令进行的，如前一小节“8.4.2 数据库备份”所述。

为了实现周期性地定时对整个数据库进行备份，需要使用操作系统的定时运行功能，指定在某个时间点运行 mysqldump 命令，例如每个周日的凌晨 2 点执行一次全库备份，因为这时用户最少，备份操作

对数据库正常运行的影响最小。

增量备份采用二进制日志实现，见下一小节“8.4.5 增量备份”的讲解。

2. 恢复过程概述

在数据库遭到损坏时（但愿永远不出现这种情况），就需要用备份的数据恢复数据库，先从全库备份中恢复数据，然后再从增量备份中恢复全库备份以后改变过的每一条数据，见表 8-5。

表 8-5 恢复策略

恢复形式	恢复的过程	恢复的内容
从全库备份中恢复	首先从最近一次的全库备份中恢复	恢复到最近一周或一天的数据
从增量备份中恢复	再从最近一次全库备份后的增量备份中恢复	恢复故障前的全部数据

从全库备份恢复是在操作系统下采用 mysql 命令进行的，如前一小节“8.4.2 数据库备份”所述。

从增量备份中恢复就是从二进制日志文件中恢复，见下一小节“8.4.5 增量备份”的讲解。

8.4.5 增量备份（二进制日志）

1. 启用增量备份

增量备份是通过二进制日志实现的，在 MySQL 的早期版本中二进制日志是默认关闭的，在 MySQL 8 中则默认是启用的，因此不再需要启用二进制日志。可以用下述命令检查二进制日志有关的配置。

```
Show variables like 'log_bin%';
```

系统变量 log_bin 的值为 ON 时，表示二进制日志是启用的，如图 8-22 所示。二进制日志保存在系统变量 log_bin_basename 所示的位置，如图 8-22 所示的位置是 C:\ProgramData\MySQL\MySQL Server 8.0\Data\，文件名是 DESKTOP-7J4QU8A-bin。

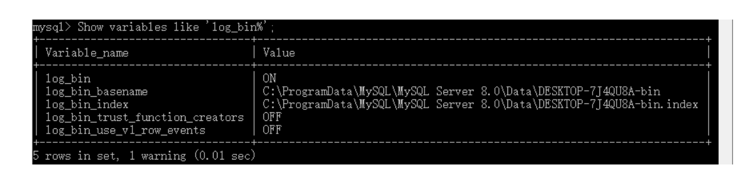


图 8-22 二进制日志相关的配置

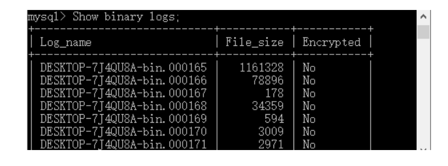


图 8-23 二进制日志文件列表

2. 二进制日志文件

二进制日志文件有许多个，其数量随 MySQL 服务器的运行而不断增加，在 MySQL 中用如下命令可以列出它们，结果如图 8-23 所示。

```
Show binary logs;
```

二进制日志文件名由三部分组成，这三部分是“计算机名”、“-bin”以及 6 个数字组成的后缀，这些文件在磁盘上如图 8-24 所示。

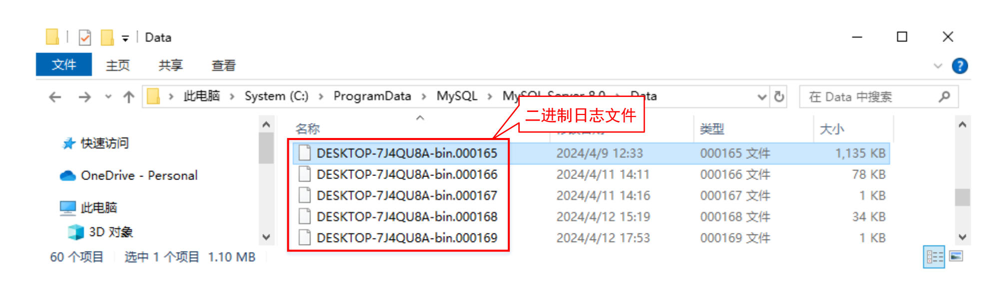


图 8-24 二进制日志文件

二进制日志文件名中的计算机名是计算机的设备名称，可以在 Windows 操作系统中“此电脑”的属性中查到，如图 8-25 所示。



图 8-25 计算机设备的名称

3. 查看二进制日志文件

在 Windows 命令行窗口，用 mysqlbinlog 命令可以查看二进制日志文件中的内容。

例如下述命令查看 DESKTOP-7J4QU8A-bin.000169 文件，部分内容如图 8-26 所示。

```
cd C:\ProgramData\MySQL\MySQL Server 8.0\Data
chcp 65001
mysqlbinlog -v DESKTOP-7J4QU8A-bin.000169
```



要在 Windows 的命令行窗口正确显示 UTF-8 字符，要用 chcp 命令改变代码页，GBK 的代码页是 936，UTF-8 的代码页为 65001，即在命令行窗口执行命令“chcp 65001”。



图 8-26 二进制日志文件中的内容

注意如图 8-26 所示的内容中如下几点。

1. 二进制日志记录下来的 SQL 语句是经过编码的，加上参数-v 可以将编码后的 SQL 语句以伪代码的形式显示出来。例如图中伪代码对应的 SQL 语句如下。

```
Update demo set age = 16 where id = 2;
```

其中的@1...@8 等对应了表的 8 个列名（id, name, sex, age, birthday, mobile, height, weight），where 条件中的数据对应修改前的行，set 赋值的数据对应修改后的行，修改前后数据有差异的只有 age 列，修改后为 16。

2. 每一个记录项都有一个位置编号，格式为“# at 位置编号”，位置编号不是连续的。例如图中所示的“# at 386”的位置编号是 386。
3. 每一个记录项都记录发生的日期时间，格式为“# yymmdd hh:mm:ss”，例如图中所示的“#240412 17:53:02”的日期时间是 2024 年 4 月 12 日 17 时 53 分 02 秒。

1) 查看某一段日期时间之间的二进制日志文件

语法格式如下。

```
mysqlbinlog --start-datetime=开始日期时间  --stop-datetime=结束日期时间  二进制日志文件
```

例如下述代码，查看 2024-04-09 10:35:00 到 2024-04-09 10:40:00 之间的日志。

```
mysqlbinlog --start-datetime="2024-04-09 10:35:00" --stop-datetime="2024-04-09 10:40:00" DESKTOP-7J4QU8A-bin.000165
```

2) 查看某一段位置编号之间的二进制日志文件

语法格式如下。

```
mysqlbinlog --start-position=开始位置编号  --stop-position=结束位置编号  二进制日志文件
```

例如下述代码，查看从位置 1028679 到 1033457 位置之间的日志。

```
mysqlbinlog --start-position=1028679 --stop-position=1033457 DESKTOP-7J4QU8A-bin.000165
```



开始位置编号(start-position)所指定的位置必须是存在的，否则会找不到开始位置，结束位置编号为空时表示到文件末尾。

4. 从增量备份中恢复

从增量备份恢复是在操作系统下采用 mysqlbinlog 和 mysql 命令实现的，从二进制日志文件的某一段恢复数据。该命令的语法如下。

```
mysqlbinlog --start-datetime=开始日期时间  --stop-datetime=结束日期时间  二进制日志文件  | mysql -u root -p
```

参数说明如下。

- 开始日期时间和结束日期时间：指定需要恢复二进制日志中哪一部分的 SQL 语句。也可以改用位置编号来指定需要恢复的数据。
- 二进制日志文件名：文件名称，可以是相对文件名或绝对文件名，后者包括所属目录的名称。
- |：要用一个文件重定向元字符“|”（竖线，键盘 Enter 键上方的键）将 mysqlbinlog 的输出作为 mysql 的输入。



常用的文件重定向元字符主要有三个：大于号（>）表示输出到文件，小于号（<）表示从文件取得输入，竖线（|）表示将前者的输出作为后者的输入。

因此，从灾难中恢复数据库的步骤如下。

1. 从最新的全库备份中恢复数据库。
2. 从增量备份中恢复全库备份以来的增量备份。

注意在恢复的整个过程中，必须严格禁止用户的访问，防止用户操作对恢复过程的干扰。在数据库完全恢复以后，才能重新允许用户的访问。

8.5 日志

8.5.1 日志概述

运维工作不仅仅是数据库安全、数据备份和恢复等，还有一项很重要的工作是监控系统的运行状况，例如运行状态、出现的错误、以及性能指标等，通过 MySQL 的日志功能能够得到一些有用的信息。

MySQL 日志系统由下述四类日志组成。

- 错误日志：记录启动、运行或停止 mysqld，以及运行时出现的各种诊断信息。
- 通用日志：记录建立的客户端连接和客户端发送到服务器的所有语句。
- 二进制日志：记录会导致数据库更改（如创建表操作或增删改操作）的 SQL 语句，主要作为数据库的增量备份使用，在上一节的“8.4.5 增量备份（二进制日志）”中已经作过讲解。
- 慢查询日志：记录执行时间超过一定时长的 SQL 语句，这样的语句是影响数据库性能的瓶颈，是进行优化的候选对象。

本节讲解错误日志和慢查询日志。

8.5.2 错误日志

Jitor 实训 | 附录 C 实训 8-5

错误日志默认是开启的，并且不能关闭。

1. 查看错误日志

错误日志文件在不同平台下，位于不同的位置，可以用下述命令查询其文件名和所在的目录。

```
Show variables like "log_error";
```

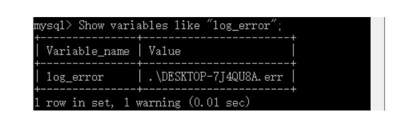


图 8-27 错误日志文件名

图 8-27 显示错误日志文件是 DESKTOP-7J4QU8A.err，其中 DESKTOP-7J4QU8A 是计算机的名字（参见图 8-25），文件名前的“.\”表示当前目录，即数据目录 C:\ProgramData\MySQL\MySQL Server 8.0\Data。

错误日志文件主要记录各种诊断信息，如服务器启动和关闭时间以及服务器运行时出现的 Error、Warning、Note 信息。可以用任意的文本编辑器打开日志文件，如图 8-28 所示是其中的部分内容。

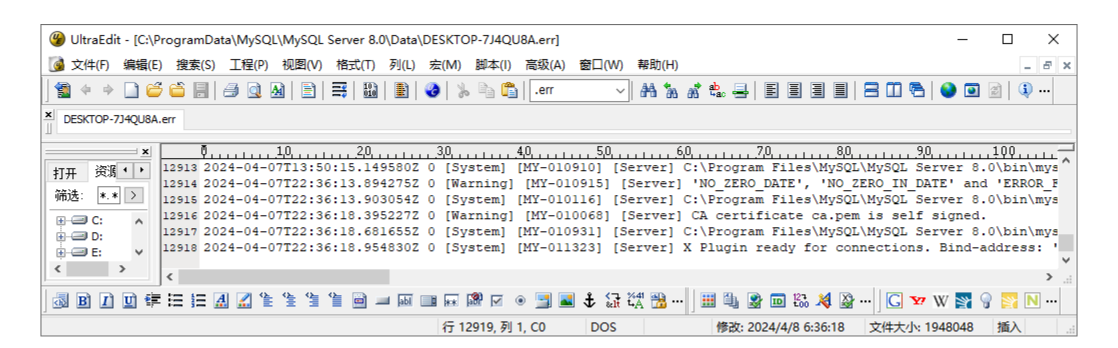


图 8-28 错误日志的部分内容

2. 用图形界面查看错误日志

可以在 Workbench 的图形界面查看错误日志，如图 8-29 所示。

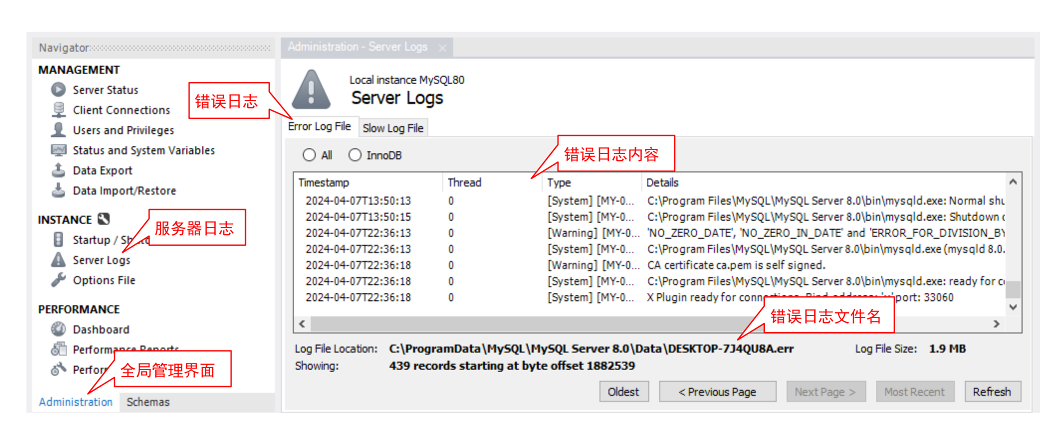


图 8-29 用图形界面查看错误日志

8.5.3 慢查询日志

慢查询日志记录执行时间超过系统变量 long_query_time 指定时间（秒）的 SQL 语句，以便找出数据库性能的瓶颈，从而对影响性能的 SQL 语句进行优化。

错误日志在 MySQL8.0 中默认是开启的，不需要时也可以关闭。

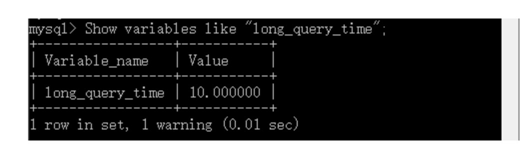
1. 配置慢查询日志

用下述语句查看慢查询日志的配置，查询结果如图 8-30 所示。

```
Show variables like "slow_query%";
```

```
mysql> Show variables like "slow_query%";
+---------------------+-------------------------------+
| Variable_name       | Value                         |
+---------------------+-------------------------------+
| slow_query_log      | ON                            |
| slow_query_log_file | DESKTOP-7J4QU8A-slow.log      |
+---------------------+-------------------------------+
2 rows in set, 1 warning (0.01 sec)
```

图 8-30 慢查询日志的配置



```
mysql> Show variables like "long_query_time";
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
1 row in set, 1 warning (0.01 sec)
```

图 8-31 慢查询日志的配置

从图 8-30 看到，慢查询日志是开启的，日志文件名是 DESKTOP-7J4QU8A-slow.log，其中的 DESKTOP-7J4QU8A 是计算机名，文件位于 C:\ProgramData\MySQL\MySQL Server 8.0\Data。

用下述语句查询慢查询日志的配置参数 long_query_time，查询结果如图 8-31 所示。

```
Show variables like "long_query_time";
```

从图 8-31 看到，系统变量 long_query_time 的值是 10，就是说慢查询日志记录所有执行时间超过 10 秒的 SQL 语句。

作为一个演示，下面将这个值改为 2 秒。打开位于 C:\ProgramData\MySQL\MySQL Server 8.0 的配置文件 my.ini，找到如下部分（中文注释是后加的），修改 long_query_time 的值为 2。

```
# General and Slow logging.
log-output=FILE
# 通用日志是禁用的（0 表示禁用）
general-log=0

general_log_file="DESKTOP-7J4QU8A.log"
# 慢日志是启用的（1 表示启用）
slow-query-log=1

slow_query_log_file="DESKTOP-7J4QU8A-slow.log"
# 将 10 秒改为 2 秒
long_query_time=2
```

修改并保存 my.ini 文件后，重启 MySQL 服务器，使这个配置生效。

2. 查看慢查询日志

作为演示，执行下述语句，其中使用了一个内置函数 sleep()，它会暂停参数所指定的时间（秒），因此下述语句执行时，每查询 1 行暂停 0.6 秒，查询 5 行的执行时间是 3 秒多一点点，超过了 2 秒的阀值。

```
Use abc;
Select *, sleep(0.6) from demo limit 5;
```

在 MySQL Workbench 中执行这条查询的信息如图 8-32 所示。

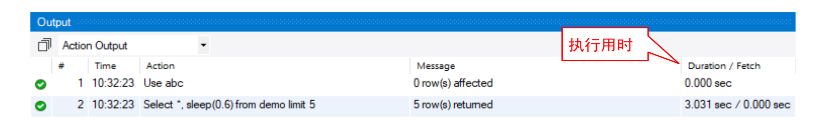


图 8-32 执行查询的信息

执行后，打开慢查询日志文件 DESKTOP-7J4QU8A-slow.log，在文件的最后，有如下日志内容。

```
# Time: 2024-04-14T02:32:16.189272Z
# User@Host: root[root] @ localhost [::1]    Id:     15
# Query_time: 3.031158   Lock_time: 0.000808 Rows_sent: 5   Rows_examined: 5
use abc;
SET timestamp=1713061933;
Select *, sleep(0.6) from demo limit 5;
```

日志中显示这条查询的执行时间是 3.031158 秒，在慢查询日志中记录下来的 SQL 语句就是可以考虑进行优化的 SQL 语句。

【案例讲解】应用项目的管理



打开附录 D 的“项目 8a 应用项目的管理”，对照下面的讲解，看看在项目中是如何进行数据库的安全管理的。

任务 1 数据库级安全

数据库级的安全措施主要是为应用项目创建一个用户账号，该账号只拥有应用项目的数据库的访问权限。对于书店管理项目，因为项目的数据库名为 mybook，因此在 MySQL 中创建一个同名的用户账号，并授予该用户访问数据库 mybook 的所有权限。

```
Create user mybook@localhost identified by '123456';
Grant all privileges on mybook.* to mybook@localhost;
```

然后配置该书店管理项目使用该用户账号访问数据库，因为书店管理项目在是 Jitor 校验器（作为 Web 服务器访问数据库）上运行的，所以是在 Jitor 校验器上“配置”，配置时需要指定数据库名、用户账号和密码，连接测试成功即可，如图 8-33 所示。



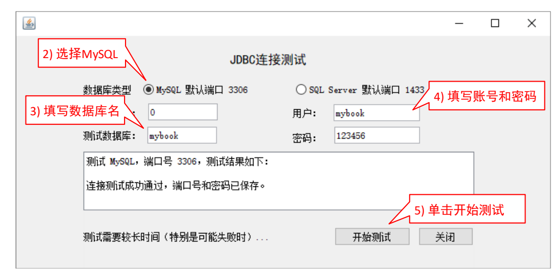


图 8-33 在 Jitor 校验器配置数据库名、用户账号和密码

配置完成后，再从浏览器打开实战项目，只要数据库是 mybook 的项目，都能正常打开，而数据库不是 mybook 的项目，例如“项目 2a “图书信息数据库”项目”，打开时弹出如图 8-34 所示的信息。

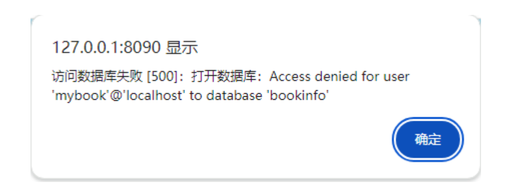


图 8-34 没有权限打开 bookinfo 数据库

上述配置提高了数据库的安全性，体现在下述两个方面。

- 只需要向应用程序提供专用的用户账号和密码，无需提供根用户密码，避免根用户密码的泄露。
- 防止一个项目访问其他数据库的数据。在这个例子中，mybook 数据库含有多个项目，在这些项目内是可以互相访问，但不能访问其他数据库的项目。

任务 2 应用项目级安全

前述的安全措施是基于数据库的，通常应用程序本身也需要一套用户账号系统，以确定应用项目中用户的身份和权限。为此，在项目数据库 mybook 中创建下述两张表（角色表和用户表）。

```
create table app_role (
	id int primary key not null auto_increment,
	col_role varchar(50) not null default ''
);
insert into app_role values (1,'总经理'),(2,'部门经理'),(5,'普通员工'),(6,'销售'),(7,'采购');

create table app_user (
	id int primary key not null auto_increment,
	col_name varchar(50) not null default '',
	col_account varchar(50) unique not null default '',   -- 唯一性约束
	col_password varchar(50) not null default '',
	col_roles varchar(50) not null default ''
);
insert into app_user values (1,'张三','zhangsan','123456','[1]'),(2,'李四','lisi','123456','[5,6]');
```

然后在“开发工具”的“通用全局设置”中填写身份认证、角色列表以及注册新用户代码，如图 8-35 所示，这三种代码是根据上述角色表和用户表编写的，读者可以根据需要加以修改。



图 8-35 身份认证、角色列表以及注册新用户代码

身份认证用于登录时的身份认证，代码如下。

```
Select id as userId, col_account as userAccount, col_name as userName, col_roles as userRoles from app_user where col_account='{$userAccount}' and col_password='{$userPassword}';
```

角色列表用于权限授予，代码如下。

```
Select id, col_role name from app_role;
```

注册新用户用于自主注册新用户，如果无需此项功能，保留此处为空。代码如下。

```
Insert into app_user (col_account , col_password) values ('{$userAccount}','{$userPassword}');
```

访问权限是通过两个步骤实现的，第一步是赋予用户角色，在“用户管理”中设置，如图 8-36 所示。第二步是指定菜单项的角色权限，在修改菜单中设置，如图 8-37 所示，其中“@所有人”是一个内置角色，表示所有登录成功的用户。

用户与角色之间是多对多联系，角色与菜单项之间也是多对多联系（这里的多对多联系没有通过表之间两个一对多来实现）。用户在登录成功时，会记录拥有的角色，而在单击菜单项时，会检查是否有权限使用该菜单项，如果没有权限，则拒绝访问。



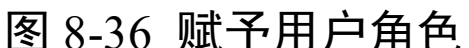


图 8-36 赋予用户角色

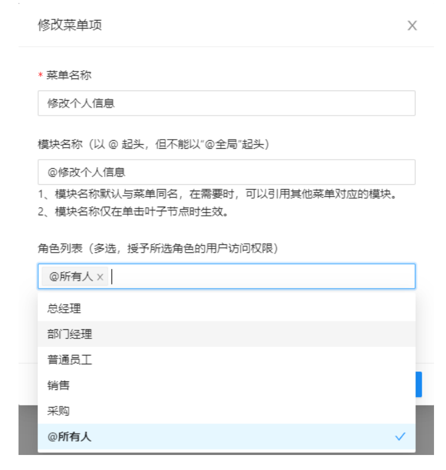


图 8-37 指定菜单项的角色权限

完成上述配置后，就可以使用应用项目自己的身份认证和权限授予功能。体现在如下方面。

- 如果还没有账号，可以在登录前注册新用户。
- 登录后，只能访问有权限访问的菜单项。

任务 3 数据备份和数据恢复

1. 数据备份

在操作系统上用 mysqldump 命令备份“项目 8a 应用项目的管理”的数据，即 mybook 数据库中的 4 张表 app_user、app_role、app_data 和 jitor_systemb8a（最后这张表是项目设计用的），备份文件名自定。



可以用 mysqldump -help 来查看 mysqldump 命令的帮助信息，用 mysqldump --help 来查看更为详细的帮助信息。

2. 数据恢复

先创建一个名为 myapp 的数据库，然后在操作系统上用 mysql 命令将上述备份文件，恢复到 myapp 数据库中，再用如下网址访问 myapp 数据库，证明数据恢复是成功的。

```
http://127.0.0.1:8090/mysql/myapp?app=b8a
```

【实战演练】图书借阅系统的管理



参考附录 D 的“项目 2d 图书借阅项目”，在读者自行开发的图书借阅项目上，完成下述操作。

- 在 MySQL 上为图书借阅项目创建一个用户账号，授权其访问图书借阅项目的数据库，并在 Jitor 校验器上配置为该账号。
- 尝试建立一套图书借阅项目身身的用户账号系统，这个系统至少需要 2 张表，并在“通用全局设置”中进行配置。

● 备份图书借阅项目的数据（全库备份），然后丢弃数据库后再从全库备份中恢复数据，或恢复到另外一个数据库中。

【单元重点】

单元小结参见本单元的【思维导图】，单元重点如下。

● MySQL 命令行的使用，理解 MySQL 命令行的优势，了解它在远程管理中的作用。
● MySQL 服务器的启动和停止、配置文件及其修改、3 种常用存储引擎的特点
● 用户管理，权限的种类和层次，权限的授予
● 备份与恢复，备份的策略（全库备份+增量备份），二进制日志的作用

【课后思考】

一、选择题

1. Windows 和 Linux 操作系统下启动 MySQL8 的命令分别是【　　　】。
   A. net start mysql80　B. net stop mysql80　C. service mysql start　D. service mysql stop
2. 支持事务和外键约束的存储引擎是（　　）。
   A. CSV　B. InnoDB　C. MEMORY　D. MyISAM
3. 完整的用户账号名称是【　　　】。
   A. limin　B. limin@%　C. limin@localhost　D. limin@25.12.23.34
4. 授予权限的命令是（　　）。
   A. grant…on…from…　B. grant…on…to…　C. revoke…on…from…　D. revoke…on…to…
5. 全库备份是在（　　）实现的。
   A. 操作系统下使用命令 mysqldump　B. 连接 MySQL 后使用命令 mysqldump
   C. 操作系统下使用命令 mysql　D. 连接 MySQL 后使用命令 mysql
6. 增量备份采用（　　）方式实现。
   A. mysqldump 命令　B. mysql 命令　C. 二进制日志　D. 慢查询日志

二、填空题

1. 使用 MySQL 命令行连接到服务器的最简单格式是__________________________________。
2. 用户授权时的三个层次是__________、___________和表、列和例程层次。

三、思考题

1. 数据库安全的目标是什么？
2. 造成数据损坏或丢失的可能原因有哪些？

【课外拓展】

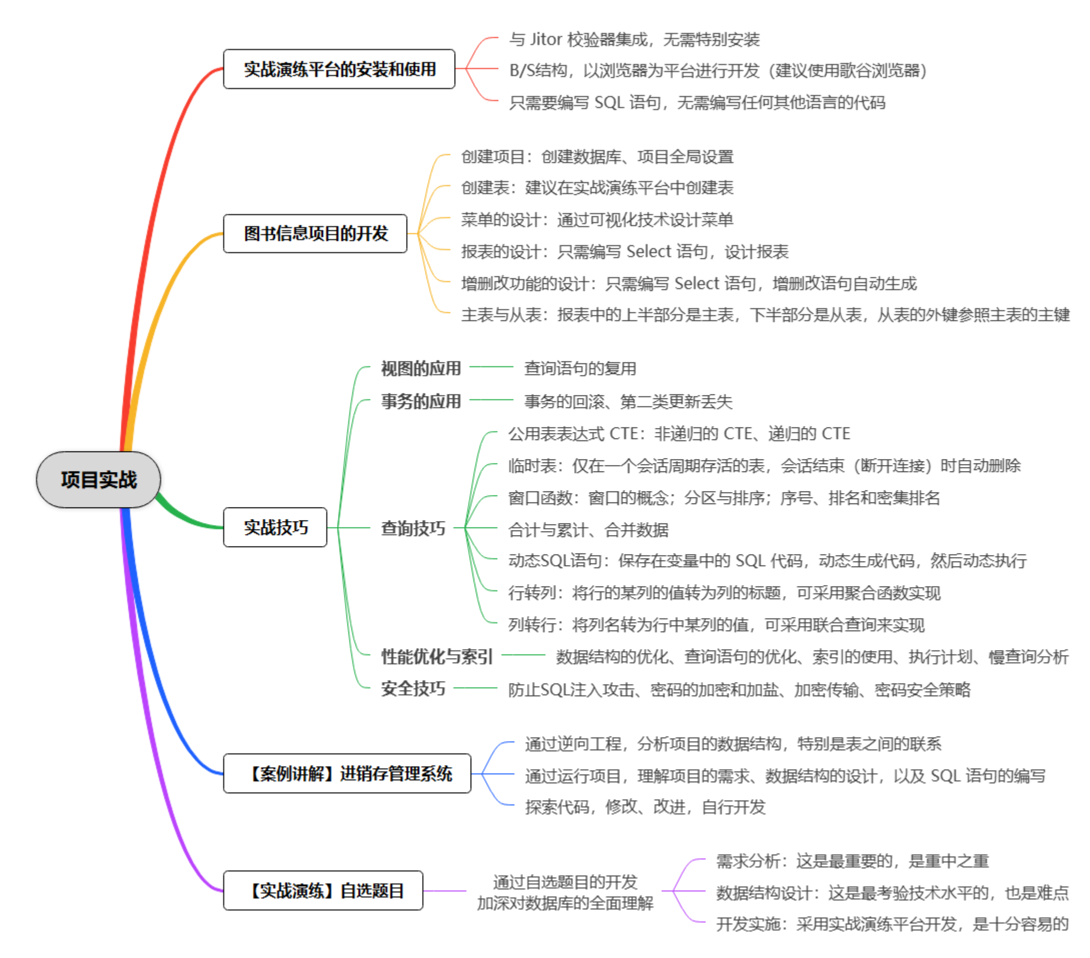
1、 阅读一篇有关数据库运维的文章，了解系统管理员的职责和运维工作的重要性。
2、 从网上查询“CSDN 泄密事件”，分析一下造成这个事件的原因和后果。

单元9 项目实战

【学习目标】

知识目标	◆ 学会 SQL 语句编写技巧。
◆ 通过运行演示项目，加深对数据库的全面理解。	◆ 初步学会使用 SQL 执行计划。
◆ 理解事务的重要性。	◆ 学会在实战演练平台上开发一个项目。
◆ 理解索引在性能优化上的应用。	**素质目标**
能力目标	◆ 创新思维、批判性思维、终身学习。
◆ 学会实战演练平台的安装和使用。	◆ 保护用户隐私、防止数据泄露、专业道德。
◆ 学会视图和事务的应用。	◆ 全局观念、道德规范、遵守法律法规。

【思维导图】



【情景导入】

小明学完了单元 1～单元 8，对 MySQL 数据库有了比较全面的认识和理解，也能够进行一些简单的开发，现在他迫不及待地想要做一个完整的项目开发，检验自己学习的成果。让我们同小明一起开发项目吧。

【知识储备】

本单元使用实战演练平台开发具体的应用项目，讲解 SQL 的应用和实战技巧，读者可以从中加深对数据库以及 SQL 语言的理解。



在实战演练平台上不需要编写任何其他语言的代码，仅使用 select 语句就能完成一个 B/S 应用项目的开发。只在需要定制功能时，才要编写 insert、update、delete 和事务处理语句。

9.1 实战演练平台的安装和使用

本节讲解实战演练平台的安装和使用，本单元所有项目都必须在这个平台上运行。

9.1.1 实战演练平台的安装

实战演练平台集成在 Jitor 校验器中，Jitor 校验器作为一个 Web 服务器，为实战演练平台提供后台服务，安装和使用方法见附录 C 的说明。Jitor 校验器启动后，打开浏览器，输入如下地址。

```
http://127.0.0.1:8090/mysql
```

9.1.2 实战演练平台的使用

实战演练平台的主界面如图 9-1 所示。横幅下是工具条，默认是“运行模式”的工具条，左侧是用户菜单，中部大块的区域是操作区，即是用户的操作区，也是开发设计的操作区。



图 9-1 实战演练平台的主界面（运行模式）

图 9-1 所示的主界面上有“实战项目列表”的链接，打开它，就能访问附录 D 中列出的所有实战项目。



每个项目都可用系统管理员 admin 或超级用户 super（密码与账号相同）登录，前者拥有开发功能，后者没有开发功能。演示版仅允许 super 登录，切换到开发版后才允许 admin 登录。

实战演练平台用于开发数据库项目，开发好的项目也必须在这个平台上运行。因此平台有两种运行模式，单击左上角“欢迎您！系统管理员”按钮将在“运行模式”和“开发模式”之间切换，如果以“超级用户”身份登录，则无法切换到开发模式。

- 运行模式：工具条上只有“报表”、“运行日志”、“帮助”和“退出”，而没有开发工具，如图 9-1 所示，这时项目处于平时运行的状态，也就是终端用户看到的界面。
- 开发模式：工具条上增加了“报表设计”、“增删改设计”和“开发工具”三组开发工具，如图

9-2 所示，这时同时拥有开发和运行能力。

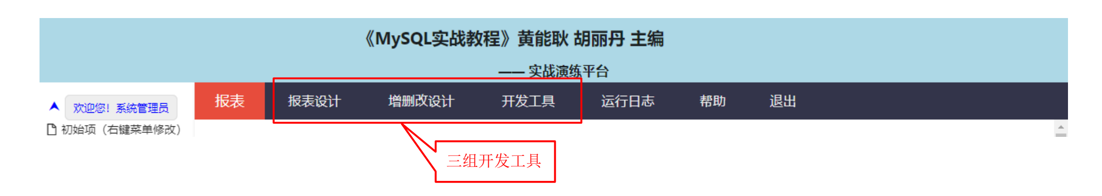


图 9-2 开发模式的界面（开发工具）

9.1.3 开发过程概述

使用实战演练平台进行项目开发的过程就是使用“报表设计”、“增删改设计”和“开发工具”这三组工具进行设计的过程，其流程如下。

1. 需求分析

任何一个项目的开发都是从需求分析开始的，参见单元 2 的有关讲解。

2. 数据结构设计

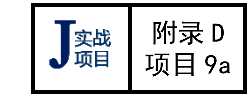
按照关系数据库的设计规范，设计数据库的数据结构，参见单元 2 的有关讲解。

3. 创建数据结构

根据数据结构设计的成果，创建数据库和创建表，表的数据结构应该与数据结构设计的要求完全一致。具体实施办法有如下三种，读者可选择其中之一来进行开发。

1. 直接编写 Create 语句，创建数据库和表，参见单元 3 的有关讲解。
2. 使用数据建模工具设计数据库，通过正向工程创建数据库和表，参见单元 3 的有关讲解。
3. 采用实战演练平台“开发工具”的“创建或修改表结构”工具来创建表。

4. 应用开发



打开附录 D 的“项目 9a 实战演练平台功能演示”，了解该平台具有哪些功能，例如图片上传和下载、级联列表等功能，然后在实际开发中使用这些功能。

第一步设计菜单：根据系统功能设计的要求设计菜单。将鼠标移到左侧的菜单文字上，再从其右键菜单中选择功能，可以修改、删除、添加菜单项，并指定关联的模块名称。

第二步设计界面：根据系统功能设计的要求，为每一个模块设计界面，实现所需的功能。单击菜单（叶子节点）将打开关联的模块，选择“报表设计”工具下的“主表设计”开始设计，设计好后要记得保存。需要时，再设计其他功能，例如从表设计、检索条件设置、分类树设计等等。

第三步测试使用：单击菜单（叶子节点），打开设计好的模块，验证第二步设计的功能，并重复第二步完善各项设计。

5. 运行维护

实战演练平台是为 MySQL 教学而设计的，不适用于实际项目，因此对项目的运行维护不作讨论。

9.2 图书信息项目的开发

在上节讲解的开发过程中，与实战演练平台相关的流程是创建数据结构和应用开发两个部分。下面以单元 2“2.2 体验关系数据库”的图书信息数据库对这两个部分进行讲解。

9.2.1 创建项目

1. 创建数据库

根据项目需求分析的结果在 MySQL 客户端上创建一个数据库。然后以下述网址打开项目。

```
http://127.0.0.1:8090/mysql/数据库
```

如果需要，也可以加上 app 标识。

一个空白的项目创建完成，但是还需对项目进行全局设置。

2. 配置全局设置

配置项目的全局设置如图 9-3 所示，按照需求分析的结果进行配置，然后单击“保存全局设置”按钮完成设置。

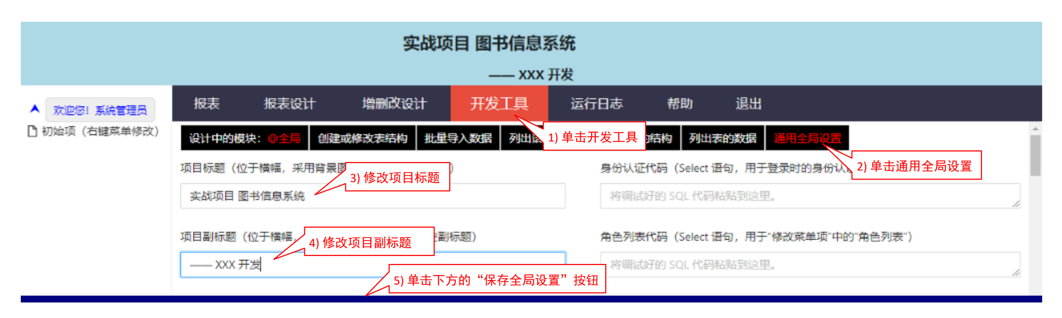
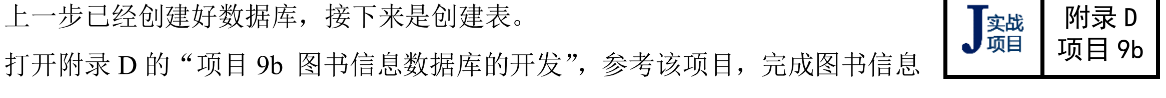


图 9-3 项目全局设置

9.2.2 创建数据结构

上一步已经创建好数据库，接下来是创建表。



打开附录 D 的“项目 9b 图书信息数据库的开发”，参考该项目，完成图书信息项目的开发。

1. 创建表结构

创建或修改表结构可以在任何 MySQL 客户端上完成，但是实战演练平台也提供了这个功能，虽然这个功能并不完善，但是可以与后续的开发过程进行很好的衔接，因此建议使用实战演练平台来创建或修改表结构。

创建表结构的过程如图 9-4 所示，该图所示的是创建“图书信息_出版社”表的界面，与单元 2 的图书信息数据库中的表相似，不同的是表名和列名采用拼音首字母。



图 9-4 “图书信息_出版社”表的设计



拼音首字母命名和英文命名各有优缺点，前者简单方便，后者可读性和可维护性好。可根据实际情况选用。

在实战演练平台中创建表，注意下述 4 个特点。

- 表名根据表标题自动生成，生成的原则是，中文表标题被转换为小写的拼音首字母，英文表标题被转换为小写，空格、减号替换为下划线。强烈建议表标题的格式为“模块名_表名”，即必须含有下划线，这个下划线有特殊作用。
- 列名根据列标题自动生成，生成的原则是，中文列标题被转换为大写的拼音首字母，英文列标题被转换为小写，因此从列名的大小写就知道列标题是中文还是英文的。列标题不允许含有下划线，因为下划线保留用于外键。
- 列的显示标题由列标题复制而来，并可修改，显示标题清空后，会根据一定的规则自动重新生成。显示标题是用于显示在界面上供终端用户阅读的标题，将被所有组件引用，而不需要多次输入。因此在界面开发过程中，不需要输入任何汉字，就能开发出完整的中文界面。
- 外键列的列标题直接使用父表的表标题（格式是“模块名_表名”），因为表标题中含有下划线，因此会被识别为外键，这时列名自动加上前缀 id_，显示标题自动加上后缀 ID，如图 9-5 所示的最后一行，“图书信息_出版社”列。



#	列标题（中英文）	列名（自动...	显示标题	类型	长度(, 精度)	输入类型	选项数据源	级联...
1	ID	id	ID	int				
2	作者	ZZ	作者	varchar	50	text		
3	书名	SM	书名	varchar	50	text		
4	出版年份	CBNF	出版年份	year		year		
5	ISBN书号	isbnsh	ISBN书号	varchar	50	text		
6	价格	JG	价格	decimal	13,2	decimal		
7	图书信息_出版社	id_tsxx_cbs	出版社ID	int		select	Select id, CBS name from tsxx_cbs; -- 修...	

图 9-5 “图书信息_图书”表的设计（示外键列的命名）

2. 修改表结构

修改表结构时，要先“选择并加载选定表的结构”，操作过程与创建表结构相同，不再赘述。

9.2.3 菜单的设计

新项目的菜单只有一个初始项。将光标移到菜单文字的上方（不能在空白处或菜单前的图标上），从其右键菜单上选择“修改”、“删除”或“添加菜单项”，如图 9-6，就能实现对菜单项的增删改。



图 9-6 菜单项的右键菜单

对菜单可以进行如下操作。

- 修改菜单项：修改菜单项的名称，并与一个同名的模块名关联起来。
- 添加菜单项：添加的新菜单项位于菜单的底部，菜单名为“新项”，然后再修改它的名称。
- 删除菜单项：删除指定的菜单项及其子菜单，删除菜单并不会删除与之关联的模块。

还可以对菜单进行排序或生成多级菜单，具体过程见“帮助”页的说明。

请读者为本项目创建 3 个菜单项，如图 9-7 左侧的菜单所示。

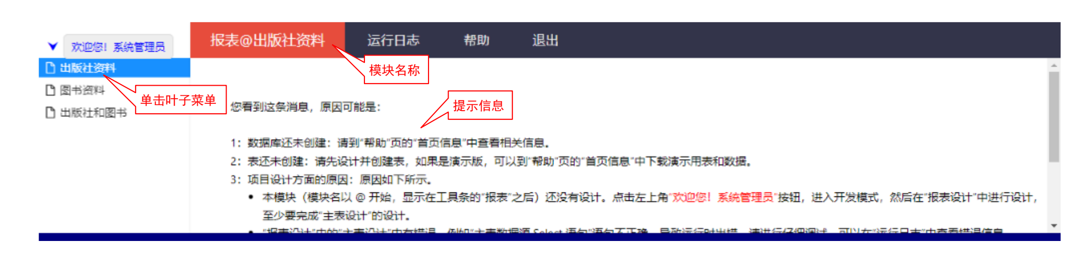


图 9-7 创建菜单

每个菜单项都有一个对应的模块，单击菜单就是打开菜单对应的模块。打开模块时，如果模块还未设计，或模块中存在错误，则会提示错误信息，否则将打开该模块供用户使用。不论是否出现错误，模块名称都会显示在工具条上的“报表”之后，模块名是以符号@起头的，很容易识别，如图 9-7 所示。

9.2.4 模块设计概述

模块是实现菜单功能的一组设计组件，这些设计组件分为两大类：报表设计和增删改设计。

1. 报表设计

报表设计用于展现数据，不能对数据进行增删改操作。报表设计有 4 个组件，如图 9-8 所示。其中“主表设计”是每个模块必须有的，其他 3 个组件都是可选的。



图 9-8 报表设计子工具条

1) 主表设计——数据源

主表设计需要一个数据源，数据源指定了将要展现的数据的来源，最简单的格式如下。

```
Select * from 主表;
```

这是一条 Select 语句，也可以是一条 Call 语句，调用存储过程得到查询结果。

2) 主表设计——展现界面

单击“提取列定义”将从数据源中提取列定义，列定义中还包括了在“开发工具”的“创建或修改表结构”中设置的“显示标题”。

在必要时，可以调整显示的顺序、显示宽度，以及否隐藏。

最后单击“保存设计”，完成主表设计后，单击菜单项，查看设计是否符合要求。然后再进行从表设计等其他组件的设计。

3) 从表设计——数据源和展现界面

如果要展现的数据是主从表的，则还需要进行“从表设计”，过程与主表设计完全相同。最重要的区别是数据源的 Select 语句应该以下述格式编写。

```
Select * from 从表 where id_外键 = {$rowId};
```

其中的{$rowId}将会被替换为用户在主表上选择的行的 id 值。

2. 增删改设计

增删改设计是在报表设计的基础上，增加增删改功能。增删改设计共有 4 个组件，只有一个组件是可选的，“主表表单设计”和“从表表格设计”分别对应报表设计中的“主表设计”和“从表设计”，对应的主表是相同的，如果有从表，则对应的从表也是相同的。用户在增删改操作之后，就能立即在报表中看

到增删改的结果。



图 9-9 增删改设计子工具条

1) 主表表单设计——数据源和表单界面

设计过程与主表设计相近，不同的有两点。一是数据源的格式如下。

```
Select * from  主表  where id={$rowId};
```

二是列的属性里多了“输入类型”等与输入界面有关的属性，例如用单选按钮、多选按钮或下拉列表等形式，通常采用默认值即可。

2) 从表表格设计——数据源和表格界面

设计过程与从表设计相近，连数据源的格式都是一样的，如下所示。

```
Select * from  从表  where id_外键  = {$rowId};
```

不同的是，列的属性里多了“输入类型”等与输入界面有关的属性，还有汇总列，通常采用默认值即可。

3) 保存数据的 SQL 代码

大多数情况下使用自动生成的核心代码即可，生成后直接保存设计。在需要时，可以修改生成的代码，例如“9.4.1 事务的回滚”和“9.7.2 密码的加密和加盐”中的例子。

如果这部分代码缺失，新增和编辑时，数据就无法保存（insert 或 update）。

4) 删除功能的 SQL 代码

大多数情况下使用自动生成的核心代码即可，生成后直接保存设计。

如果这部分代码缺失，删除功能就无法实现。

9.2.5 出版社资料模块的设计

本小节对“出版社资料”菜单项的“@出版社资料”模块进行设计。

1. 设计出版社表的报表

1) 设计报表

创建报表的过程如图 9-10 所示，其中“主表数据源 Select 语句”要改为如下。

```
Select * from tsxx_cbs;              -- 语句结束一定要加分号，注释符 -- 之后一定要有一个空格
```



图 9-10 设计报表

主表数据源 Select 语句应先在 MySQL Workbench 调试通过，并得到正确的查询结果，其中的表名 tsxx_cbs 是表标题“图书信息_出版社”的拼音首字母，在创建表时根据表标题自动生成的。

图 9-10 中显示标题的中文是从创建表时的显示标题提取而来的，通常情况下不需要修改。

完成后单击“保存设计”。至此，“出版社资料”模块的主表设计完成。

2) 输入测试数据

为了测试上述设计结果，可以在 MySQL Workbench 向出版社表输入一到两行测试数据。

3) 查看报表

假设输入了如图 9-11 所示的数据，就可以通过刚才设计的报表来查看表中的数据。单击菜单项“出版社资料”，运行结果如图 9-11 所示。



图 9-11 “查询出版社”的运行结果

报表的上方有“新增”、“编辑”、“删除”和“查看”4 个按钮，目前还没有为它们设计功能，因此单击它们会出错。如果不需要这些功能，也可以在设计界面中隐藏它们，以免误导用户。

4) 查看运行日志

单击工具条上的“运行日志”，可以看到运行过程中后台运行的 SQL 语句，如图 9-12 所示。



图 9-12 运行日志

2. 设计出版社表的增删改

1) 设计主表表单



图 9-13 主表表单设计

上一小节完成了主表报表的设计，现在要设计主表表单。表单是用于“新增”和“编辑”的，它们采用相同的界面，只是在后台处理方面有些区别，“主表表单”的设计界面如图 9-13 所示。

图 9-13 的“主表编辑 Select 语句”代码如下。

```
Select * from tsxx_cbs where id={$rowId};
```

其中{$rowId}将被替换为用户想要编辑的行的 id 值，如果是“新增”，则替换为 0。

完成后单击“保存设计”。至此，主表表单设计完成。

2) 编写保存数据的 SQL 语句

主表表单设计完成，还要编写保存数据的 SQL 语句，“新增”和“编辑”都是采用相同的代码来保存数据，只是在内部会查询要保存的数据是否已经存在，如果已经存在，则用 Update 语句，如果不存在，则用 Insert 语句。操作过程如图 9-14 所示。



图 9-14 编写保存数据的 SQL 语句

至此，保存数据的 SQL 语句编写完成，代码全部是自动生成的，并不需要自己编写或修改。

3) 录入出版社数据

完成增删改的设计后，开始输入数据，单击菜单项“出版社资料”，单击“编辑”时会打开一个表单，表单中已有原来的数据，因此在单击“编辑”按钮前，要先选择想要编辑的那一行。编辑数据的界面如图 9-15 所示。



图 9-15 输入数据（编辑，已修改数据）

如果是“新增”，则会打开一个空的表单让用户输入数据，如图 9-16 所示。



图 9-16 输入数据（新增，已输入数据）

单击“保存所有数据”，就会运行“保存数据的 SQL 语句”，执行更新或插入语句。编辑第一行数据，

新增第四行数据后，报表中的数据如图 9-17 所示。



图 9-17 编辑和新增数据之后

4) 查看运行日志

查看运行日志，如图 9-18 所示，并与如图 9-17 所示的出版社表数据进行比对。



图 9-18 对应的运行日志

在图 9-18 中，可以看到编辑功能对应的是 update 语句，新增功能对应的是 insert 语句，这是在内部自动判断和处理的。

因此，在使用实战演练平台时，应该经常查看运行日志，以便熟悉后台处理的细节，加深对 SQL 的理解。

5) 删除功能的设计

删除功能的实现是很简单的，在“增删改设计”中的“删除功能 SQL 代码”编写，直接采用自动生成的代码即可，无需修改。

删除功能没有自己的界面，只是弹出一个删除确认对话框，无需设计。

9.2.6 图书资料模块的设计

本小节请读者对“图书资料”菜单项的“@图书资料”模块进行设计。

1. 设计图书表的报表

请读者参照上一小节的报表设计过程，设计“@图书资料”模块，完成图书表的报表设计。

由于图书表有一个外键“出版社 ID”是参照出版社表的（参见图 9-5 中最后一行），如果还要显示出版社名称，则修改主表数据源 Select 语句为如下代码（在 MySQL Workbench 调试后再复制过来）。

```
Select tsxx_ts.*, tsxx_cbs.CBS
    from tsxx_ts
        join tsxx_cbs on tsxx_ts.id_tsxx_cbs = tsxx_cbs.id;
```

完成后的查询图书界面如图 9-19 所示。



图 9-19 查询图书界面

2. 设计图书表的增删改

请读者参照上一小节的增删改设计过程，设计“@图书资料”模块的增删改功能，完成图书表的设计。

由于图书表有一个外键“出版社 ID”是参照出版社表的，它的值必须是出版社表中已有的主键值中的一个，需要从一个下拉列表中选择，如图 9-20 所示。



图 9-20 输入图书数据模块的编辑界面

因此在如图 9-21 所示的主表表单设计中，出版社 ID 列的输入类型应该改为 select（这个 select 不是 SQL 的关键字，而是网页中表示下拉列表的一种输入类型），并为下拉列表指定一个数据源，代码如下。

```
Select id, CBS name from tsxx_cbs;          -- 查询的结果将用于图 9-20 的下拉列表中
```



图 9-21 为出版社 ID 列指定输入类型为下拉列表，并编写列表数据源

保存设计后，输入图书数据模块的编辑界面如图 9-20 所示，其中出版社 ID 列下拉列表的数据是来自于下拉列表数据源的查询结果。

9.2.7 出版社和图书模块的主从表设计

在前述的报表设计中，完成了出版社资料模块的设计，并由读者完成了图书资料模块的设计，这样就完成了整个图书信息项目的开发。

还可以用另外一种思路来完成这个项目，将图书表和出版社表合并在一个模块内来处理。主表是出版社表，从表是图书表，两者之间是一对多的联系，这就是“出版社和图书”模块将要完成的任务。

1. 复制“出版社资料”模块

“出版社和图书”模块与“出版社资料”模块在主表方面的完全相同的，因此，将“出版社资料”模块复制为“出版社和图书”模块，如图 9-22 所示。复制后，两个模块是完全相相同的，下面将在复制出来的“出版社和图书”模块上增加从表功能，从而完成主从表的设计。

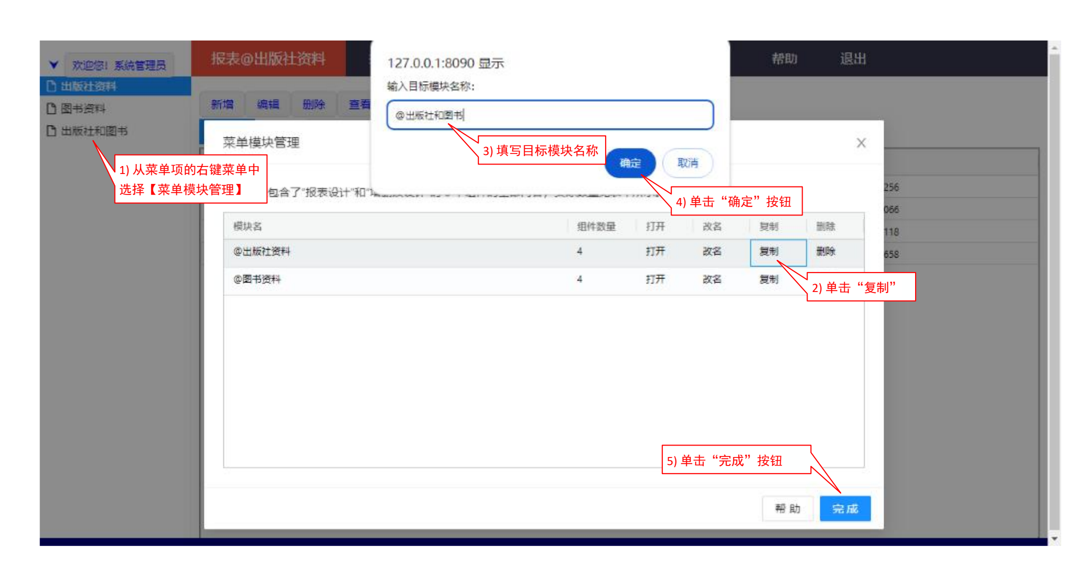


图 9-22 复制模块

2. 设计主从表报表

1) 从表设计

由于“@出版社和图书”模块的主表设计是从“@出版社资料”复制过来的，无需修改即可直接使用，只要增加一个从表设计即可，如图 9-23 所示。



图 9-23 从表设计的界面

图 9-23 的从表数据库 Select 语句，代码如下。

```
Select * from tsxx_ts where id_tsxx_cbs = {$rowId};
```

其中的 where 子句表达的意思是从表的外键等于主表的主键。保存后，从表设计完成。

2) 主从表的报表界面

运行结果如图 9-24 所示，报表的上半部分显示主表数据，下半部分显示从表数据，在主表中选中不

同的行，从表显示主表对应的数据。这样就实现了在一个报表内同时显示主表和从表的数据。



图 9-24 主从表报表的界面

3) 查看查询时的运行日志

在主表中选择不同的行，后台运行的代码如图 9-25 所示，每单击一次主表的行，就会执行一次从表数据源 select 语句，该语句如前所述。

从图 9-25 可以看到，连续执行了 3 次从表数据源 select 语句，分别对应主表的第 1、2、1 行。



图 9-25 运行日志中记录的执行从表数据源 select 语句

3. 设计主从表的增删改

1) 从表表格设计

打开“出版社和图书”模块，在“增删改设计”中设计，由于主表设计是从“出版社资料”复制过来的，这里只要进行“从表表格设计”，如图 9-26 所示。



图 9-26 从表表格设计的界面

从表中参照主表的外键必须隐藏，因为平台会自动处理从表与主表的参照。但如果是参照其他表的外键，则需要将输入类型改为 select（网页的下拉列表输入类型），并且编写列表数据源的 select 语句。

2) 主从表的保存数据 SQL 代码

至此从表表格设计完成，还有一件要做的事是编写“保存数据的 SQL 代码”，与前一小节一样，只需要直接采用自动生成的代码，并不需要作任何修改，保存即可，不再赘述。可以参见图 9-14，不同的是生成出来的代码包含了从表的部分。



只有在分别完成“主表设计”和“从表表格设计”之后，才能生成“保存数据的 SQL 代码”，因为生成时会根据是否存在“从表表格设计”而生成不同的 SQL 代码。

3) 主从表的编辑界面

完成了主从表的“主表表单设计”和编写“保存数据的 SQL 代码”之后，打开“@出版社和图书”模块，可以使用“新增”或“编辑”功能，这里以“编辑”功能来讲解，如图 9-27 所示。



图 9-27 主从表的编辑界面

在图 9-27 中，操作区的上半部分是主表的表单，用于编辑主表当前行的数据，下半部分是从表的表格，这是可编辑的表格，单击单元格可以直接修改数据，还有两个按钮，“插入新行”按钮用于向从表插入新行，新行的外键值自动填入主表当前行的主键值，“删除该行”按钮用于删除从表中选中的当前行，删除从表中的行并不会删除主表的当前行，在“保存所有数据”之前，还可以恢复被删除的行。

4) 查看保存时的运行日志

在图 9-27 的界面上单击“保存所有数据”，后台运行的代码如图 9-28 所示。



图 9-28 保存主从表数据时运行的代码

如图 9-28 所示的 4 条 DML 语句的作用分别如下，并与如图 9-27 所示的编辑界面数据进行比对。

- 更新主表当前行，用的是 update 语句。
- 删除从表第 1 行，这一行是原来的行，但要删除掉，使用 delete 语句。
- 更新从表第 2 行，这一行是原来的行，使用 update 语句。
- 插入从表第 3 行，这一行是新增的，因此用 insert 语句。

5) 主从表的删除数据 SQL 代码

在“增删改设计”的“删除功能 SQL 代码”，直接采用自动生成的代码即可，无需修改。生成时，会根据是否存在“从表表格设计”而生成不同的代码。

至此，图书信息项目的设计全部完成，并且是采用了两种方式来完成的。

9.3 视图的应用

从本节开始，对单元 5～单元 8 的部分主题作一些深入的讲解，这些部分就是实战中使用到的一些技巧，读者可自由选择学习。



单元 6 的“6.2 视图”讲解了视图，而视图在项目中应用的重要性并未得到体现。本节和下一节将通过一个实战项目来讲解视图的应用。这个项目是“项目 9c 视图的应用和事务的应用”，如图 9-29 所示。



图 9-29 项目 9c 视图的应用和事务的应用

在这个“进销存项目（待改进）”中，“库存查询”模块的“主表数据源 Select 语句”代码如下。

```
Select id, name, sum(purchase) purchase, sum(order1) order1, (sum(purchase) - sum(order1)) inventory
	from
		(Select trans_goods.id, trans_goods.name, sum(quantity) purchase, 0 order1
			from trans_purchase_detail
				join trans_goods on trans_purchase_detail.id_trans_goods = trans_goods.id
			group by trans_goods.id
		union
		Select trans_goods.id, trans_goods.name, 0 purchase, sum(quantity) order1
			from trans_order_detail
				join trans_goods on trans_order_detail.id_trans_goods = trans_goods.id
			group by trans_goods.id
		) a
group by id, name;
```

将这个查询创建为一个名为 v_inventory 的视图，代码如下。

```
Create view v_inventory as
 -- 上述库存查询的完整代码（直接复制）
```

在“库存查询（视图版）”中，主表数据源 select 语句简化为如下。

```
Select * from v_inventory;
```

这个视图还可以在下一节的“9.4.1 事务的回滚”中使用，因此视图可以在许多场合使用，体现出很好的复用性。

9.4 事务的应用

本节以两个实例来进一步讲解事务，同样是在“项目 9c 视图的应用和事务的应用”项目中。

9.4.1 事务的回滚

1) 问题的提出

这是进销存项目中销售订单的例子，在待改进版中的“销售订单”模块中，没有为销售订单可能出现的缺货情况进行处理，在缺货时仍然可以生成销售订单，这样就会给实际销售工作带来严重后果。

为了避免下单后出现缺货的情况发生，应该在下单前检测库存数量是否满足销售的要求，这个检测可以在如下几个时间点进行。

- 在销售订单添加商品时，检测该商品还有多少库存。
- 在修改商品的数量时，检测该商品的库存数量是否足够。
- 在保存销售订单之前，检测销售订单中每一种商品的库存数量是否足够。
- 保存销售订单时，开启事务，在事务中生成订单，插入或更新商品的销售数量，这时还未提交，然后检测销售订单中每一种商品的库存数量是否足够，只要有一种商品的数量不够，就要回滚事务，订单保存失败。如果订单中所有商品的库存都足够，则提交事务，订单成功生成。

前三个时间点都有缺点，因为开单的不是只有一个销售员，而是多个用户同时操作。只有最后一个时间点是万无一失的，这是事务的 ACID 特性的一个体现。

2) 解决方案

原来保存数据的代码如下（由实战演练平台自动生成）。

```
-- 包含主表和从表的代码如下：
E1: Begin
Start transaction;
Call p_jitor_saver(@err, 'trans_order', '{$rowData}', @id, '', ''); -- 第二个参数是 trans_order，最后两个参数必须为空串，其中 @id 传主表的主键值给从表
If @err is not null then
  Select '保存主表时出错（回滚）' msg, @err error;   -- 返回错误信息
    -- 存储过程中已经回滚
    Leave E1;
End if;
Call p_jitor_saver(@err, 'trans_order_detail', '{$childData}', @id, 'id_trans_order', 'trans_order'); -- 第二个参数是trans_order_detail，最后两个参数指定参照关系（注意检查外键的名称是否正确）
If @err is not null then
  Select '保存从表时出错（回滚）' msg, @err error;   -- 返回错误信息
    -- 存储过程中已经回滚
    Leave E1;
End if;
-- 需要时，可以回滚，并返回出错信息：Select '自定义出错信息' msg, 'E98009' error;
Commit;
Select '成功保存主表和从表' msg, '' error;
End;
```

上述代码是一个事务，在事务中用下述代码替换 Commit 这一行代码（倒数第 3 行，加粗斜体显示）。

```
Select group_concat(name) into @sp_list      -- group_concat 函数参见“9.5.5 合并数据”中的“3. 合并多行数据”
    from v_inventory                          -- v_inventory 见上一节的视图
        join trans_order_detail on v_inventory.id = trans_order_detail.id_trans_goods
    where inventory < 0;
if @sp_list is not null then
    Select concat(@sp_list, ' 的库存不够') msg, 'E98009' error;        -- 返回出错的原因
    Rollback;                                                           -- 库存不够时回滚
```

```
Else
        Commit;
        Select '成功保存主表和从表' msg, '' error;
End if;
```

其中使用了上一节创建的视图 v_inventory，这个视图为代码的编写调试都提供了很大的方便。

在原来保存数据 SQL 代码中，提交是无条件的，因此缺货时也能成功生成订单。

修改后的保存数据 SQL 代码中，提交之前，需要检查是否出现了库存为负数的情况，即生成订单后是否导致缺货，缺货时必须回滚，并给出相应的出错信息，否则提交销售订单的数据，订单成功完成。这样就成功地避免了缺货订单的产生。

读者可以运行项目 9c 中“销售订单（事务版）”来体验事务的回滚，修改超出库存的销售数量，并且可以切换到开发版，查看“保存数据的 SQL 代码”。



9.4.2 第二类更新丢失

在单元 7“7.7.3 锁”的表 7-9 列出了 5 种并发问题，其中最不严重的是第二类更新丢失，在许多应用程序中对此并不作任何处理，但在有些情况下，项目的需求方会要求不能发生第二类更新丢失，以免因此而导致纠纷。

1) 问题的提出

本小节以一个请假申请的审批流程来讲解。有多个审批者有权对请假进行审批（多用户环境），第二类更新丢失出现是由于两个审批者在同一时间进行了如下操作而产生的，如图 9-30 所示，后提交的数据在没有警告的情况下覆盖了先提交的数据，这时先提交的数据就丢失了。

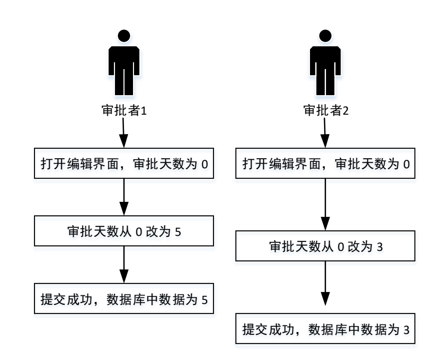


图 9-30 出现第二类更新丢失的原因

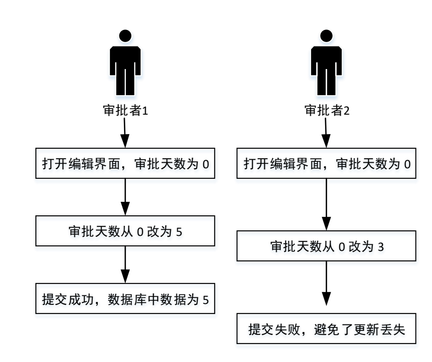


图 9-31 避免第二类更新丢失的思路

通常第二类更新丢失并不会造成多大的后果，但是如果业务流程有严格的要求，必须跟踪审批过程的细节，这时审批者 1 就会发现批准了 5 天假，却无缘无故的丢失了，变成了 3 天，同时审批者 2 会辩解说，他只是将审批天数从 0 改为 3 天，并不知道有人已经改为 5 天了。

2) 解决方案

解决的思路是在出现第二类更新丢失时，拒绝后来提交的数据（即审批者 2 的数据），如图 9-31 所示。实施的办法有多种，其中的一种办法是在表结构中添加一个名为“行版本号”的列，该列的值在每次更新时将递增 1，在更新时需要检查行版本号是否仍为原来的值，如果不是，表明有其他用户更新过，这时不允许更新，避免覆盖其他用户所作的更新，从而避免了丢失，如图 9-32 所示。

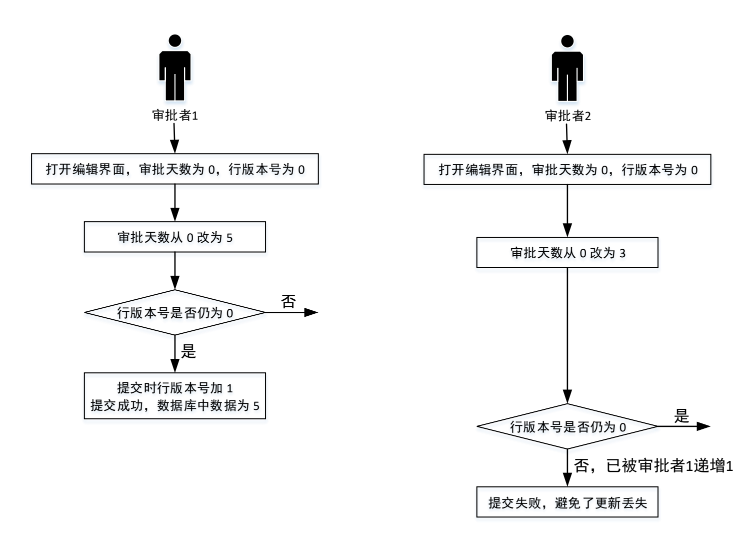


图 9-32 避免第二类更新丢失的实施办法

实战演练平台已经内置了这个功能，大多数情况下不需要这个功能，因此默认没有启用。

启用的方式是在表中添加一列名为 Version 的内置列（本平台的内置列），其余设计无需作任何改动，如果出现第二类更新丢失时，将回滚后来提交的数据，并弹出如图 9-33 所示的警告信息。

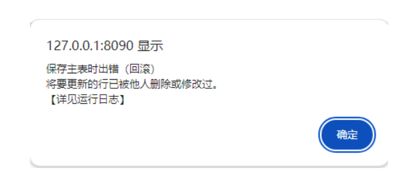


图 9-33 出现第二类更新丢失时的警告信息

在增加了避免第二类更新丢失的措施后，如图 9-30 所示的情况就再也不会发生，审批者 2 不能直接将 0 改为 3 天，因为提交失败了。如果审批者 2 确实想改为 3 天，必须再次编辑，这次编辑是明确地从 5 天改为 3 天，从而避免了纠纷的发生。

J 实战项目 | 附录 D 项目 9c

在实战演练平台上打开项目 9c，先体验第二类更新丢失。办法是打开两个浏览器窗口，打开“第二类更新丢失的原因”模块，分别模拟审批者 1 和审批者 2，按图 9-30 所示的时间次序进行操作，这时审批者 1 和审批者 2 都能正常更新，审批者 2 的更新覆盖了审批者 1 的更新，这就是第二类更新丢失。

作为比较，然后体验避免第二类更新丢失。办法也是打开两个浏览器窗口，打开“避免第二类更新丢失”模块，分别模拟审批者 1 和审批者 2，同样按图 9-30 所示的时间次序进行操作，这时审批者 1 正常更新，审批者 2 提交的数据被回滚，得到如图 9-33 所示的警告信息，避免了第二类更新丢失，如图 9-31 所示。

还可以通过查看运行日志的代码，比较这两次模拟操作过程中，后台运行的代码有什么不同，这样才能更好地理解这个问题。



“第二类更新丢失的原因”和“避免第二类更新丢失”模块的区别在于两个模块所用的表结构不同，后者的表结构中多了一个名为 version 的列，其作用就是避免第二类更新丢失。

9.5 查询技巧

本节讲解实战项目中可能用到的一些基本技巧，更多技巧见实战项目中的说明。

9.5.1 公用表表达式 CTE



单元 6“6.1 子查询”所述的子查询是嵌套查询（简单子查询和相关子查询），本小节讲解一种递归查询，这种查询采用公用表表达式（Common table expression，CTE），它是 MySQL 8 推出的新功能。

公用表表达式可以看作是临时表，但是只在一条语句内有效。在一条语句内，CTE 可以被多次引用，CTE 也可以有多个。

1. 非递归的 CTE

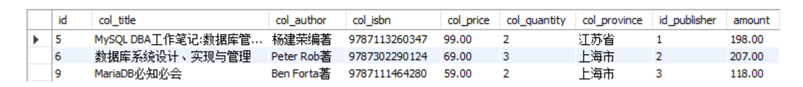
可以把作为一张表的子查询改写为 CTE。例如“6.1.1 简单子查询”中的下述子查询代码。

```
Select *
	from (Select *,
			col_price*col_quantity amount
		from book
	) a
	where amount > 100;
```

其中的子查询也称为派生表，可以将派生表改写为公用表表达式，代码如下。

```
With cte_1 as (Select *,
			col_price*col_quantity amount
		from book
	)
Select *
	from cte_1
	where amount > 100;
```

运行结果如图 9-34 所示，该结果与前面子查询的运行结果是相同的。上述代码中的 cte_1 是公用表表达式的名字，在一条语句的开始用关键字 with 定义了 cte_1，然后在这条语句的后面就能引用它。



id	col_title	col_author	col_isbn	col_price	col_quantity	col_province	id_publisher	amount
5	MySQL DBA工作笔记:数据库管...	杨建荣编著	9787113260347	99.00	2	江苏省	1	198.00
6	数据库系统设计、实现与管理	Peter Rob著	9787302290124	69.00	3	上海市	2	207.00
9	MariaDB必知必会	Ben Forta著	9787111464280	59.00	2	上海市	3	118.00

图 9-34 公用表表达式的运行结果

派生表（作为一张表的子查询）与 CTE 的不同在于如下几点。

- 派生表只能被引用一次，而 CTE 可以被引用多次。
- 一个 CTE 可以引用另一个已被定义的 CTE。
- CTE 可以是自引用的（递归的），见下面“2. 递归的 CTE”的讲解。
- CTE 可读性比派生表更好。

例如下述语句定义了两个 CTE，其中 cte_2 还引用了 cte_1，运行结果如图 9-35 所示。

```
With cte_1 as (Select *,
			col_price*col_quantity amount
		from book
	),
	cte_2 as	(Select avg(amount) a from cte_1
		where amount > 100
	)
Select cte_1.*
	from cte_1
	where amount >
		(Select a from cte_2
		);
```

id	col_title	col_author	col_isbn	col_price	col_quantity	col_province	id_publisher	amount
5	MySQL DBA工作笔记:数据库管...	杨建荣编著	9787113260347	99.00	2	江苏省	1	198.00
6	数据库系统设计、实现与管理	Peter Rob著	9787302290124	69.00	3	上海市	2	207.00

图 9-35 两个 CTE 的运行结果

2. 递归的 CTE

递归是 CTE 的一个特色，也比较难理解，初学者仅了解一下即可。这个例子的数据是如图 9-36 所示的集团公司的组织结构图，演示数据如图 9-37 所示，其中 pid 是外键，参照本表的主键，创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”。

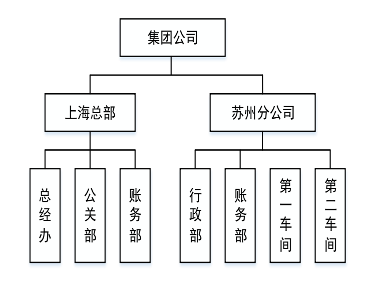
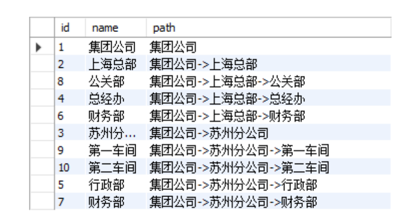


图 9-36 组织结构图



id	name	pid
1	集团公司	NULL
2	上海总部	1
3	苏州分公司	1
4	总经办	2
5	行政部	3
6	财务部	2
7	财务部	3
8	公关部	2
9	第一车间	3
10	第二车间	3
NULL	NULL	NULL

图 9-37 groupinfo 表的演示数据



id	name	path
1	集团公司	集团公司
2	上海总部	集团公司->上海总部
8	公关部	集团公司->上海总部->公关部
4	总经办	集团公司->上海总部->总经办
6	财务部	集团公司->上海总部->财务部
3	苏州分...	集团公司->苏州分公司
9	第一车间	集团公司->苏州分公司->第一车间
10	第二车间	集团公司->苏州分公司->第二车间
5	行政部	集团公司->苏州分公司->行政部
7	财务部	集团公司->苏州分公司->财务部

图 9-38 CTE 查询得到的组织结构图

编写下述语句，采用公用表表达式实现对组织结构的递归访问，查询结果如图 9-38 所示。

```
With recursive cte_1 as (select id, name, name as path
          from groupinfo
          where pid is null
     union all
          select groupinfo.id, groupinfo.name, concat(cte_1.path, '->', groupinfo.name)
          from groupinfo join cte_1
               on groupinfo.pid = cte_1.id
     )
select * from cte_1 order by path;
```

在上述代码中，别名为 cte_1 的公用表在其定义内被引用，形成了公用表的递归引用。

编写递归公用表表达式要注意以下几点。

- 必须在 with 后加上关键字 recursive。
- 公用表的定义必须是一个联合查询。
- 联合查询中的第一个查询不能引用自身的公用表，是非递归的，这个查询判断递归的结束，例子代码中的结束条件是 pid is null。
- 联合查询中的第二个查询才能引用自身的公用表，是递归的。

9.5.2 临时表



上一小节的公用表可以看作是临时表，只在一条语句内有效。但是 CTE 并不是临时表，MySQL 确实提供了临时表。临时表与 CTE 在某些方面相似，但是有较长的生命周期，可以用临时表替代 CTE，例如上一小节的第一个 CTE 例子改为临时表，代码如下所示，运行结果与图 9-34 完全相同。就是说，采用派生表、CTE 和临时表都能得到相同的结果。

```
Drop table if exists tmp_1;
Create temporary table tmp_1 as
     Select *,
               col_price*col_quantity amount
          from book;

Select *
```

```
from tmp_1
where amount > 100;
```

临时表用于保存一些临时数据，临时表与普通表的区别在于临时表只在当前连接（也称会话）可见，当关闭连接时，MySQL 会自动删除临时表。除此之外，临时表与普通表是相同的，可以使用 create、alter、drop、insert、update、delete、select 等所有操作，但是 show tables 命令不能列出临时表。



一个连接的时间可能很长（数小时），例如从 Workbench 连接到服务器，直到退出的很长一段时间，也可能很短（小于 1 秒），例如在实战演练平台上每一次单击按钮都是一个连接。

注意以下几点。

- 创建临时表时要加上 temporary 关键字，即 create temporary 表名(……)。
- 临时表只在当前连接（即会话）可见。
- 关闭连接时会自动丢弃所有临时表。
- 通常在创建前要先丢弃它，但用完之后不必丢弃它。

9.5.3 窗口函数

1. 窗口函数和窗口



讲解窗口函数之前，要先理解窗口概念。

1) 窗口

窗口是一个虚拟的概念，它定义了函数应用的数据范围。这个范围可以基于行、分区或排序来确定。窗口函数的输出是相对于这个“窗口”中的行计算出来的。

窗口函数应用在窗口上，就像从房子的外面通过窗口观察房内的情况，窗口观察到的只是房内的部分数据，即窗口所定义的数据范围。

窗口由 over()子句定义，也可以由 window 关键字定义窗口，并起一个别名，以便窗口的复用。

窗口通过“分区”（partition）定义行在窗口中的分组，或者通过“排序”（order by）定义行在窗口及其分组中的排序，或者通过“帧”（frame）定义窗口及其分组中行的子集，这 3 项都是可选的。

2) 窗口函数

窗口函数是可以在窗口中使用的函数。例如序号 row_number()函数、排名 rank()函数和密集排名 dense_rank()函数等，大多数聚合函数（如 sum()函数）同时也属于窗口函数。

2. 窗口函数的使用

1) 分区

使用窗口分区，可以对不同分区的数据进行处理。本节使用图 9-39 a)所示的成绩表原始数据，创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”。

id	name	course	score
1	张三	语文	95
2	张三	数学	90
5	李四	语文	93
6	李四	数学	81
7	王五	语文	92
10	赵六	语文	95
12	赵六	数学	88
15	孙七	语文	86
16	孙七	数学	82
19	钱八	语文	93
20	钱八	数学	88
NULL	NULL	NULL	NULL

a) 原始数据

id	name	course	score	全部总分
1	张三	语文	95	983
2	张三	数学	90	983
5	李四	语文	93	983
6	李四	数学	81	983
7	王五	语文	92	983
10	赵六	语文	95	983
12	赵六	数学	88	983
15	孙七	语文	86	983
16	孙七	数学	82	983
19	钱八	语文	93	983
20	钱八	数学	88	983

b) 无分区

id	name	course	score	学生总分
15	孙七	语文	86	168
16	孙七	数学	82	168
1	张三	语文	95	185
2	张三	数学	90	185
5	李四	语文	93	174
6	李四	数学	81	174
7	王五	语文	92	92
10	赵六	语文	95	183
12	赵六	数学	88	183
19	钱八	语文	93	181
20	钱八	数学	88	181

c) 按学生分区

id	name	course	score	课程总分
2	张三	数学	90	429
6	李四	数学	81	429
12	赵六	数学	88	429
16	孙七	数学	82	429
20	钱八	数学	88	429
1	张三	语文	95	554
5	李四	语文	93	554
7	王五	语文	92	554
10	赵六	语文	95	554
15	孙七	语文	86	554
19	钱八	语文	93	554

d) 按课程分区

图 9-39 成绩数据的窗口分区处理

将所有数据作为一个分区，这时可以统计全部总分。代码如下，结果如图 9-39 b)所示。

```
select id, name, course, score,
       sum(score) over() as '全部总分'
from student;
```

其中 over 子句的括号内是一个窗口，括号内容为空，表示所有数据作为一个分区，也没有排序，窗口函数要与窗口一起使用。

将所有数据以学生姓名进行分区，这时可以统计学生总分。代码如下，结果如图 9-39 c)所示。

```
select id, name, course, score,
       sum(score) over(partition by name) as '学生总分'
from student;
```

将所有数据以课程名进行分区，这时可以统计课程总分。代码如下，结果如图 9-39 d)所示。

```
select id, name, course, score,
       sum(score) over(partition by course) as '课程总分'
from student;
```

用 window 子句为窗口起一个别名，上述代码改为如下述代码，结果与图 9-39 d)完全相同。

```
select id, name, course, score,
       sum(score) over w as '课程总分'              -- 使用窗口 w
from student
       window w as (partition by course);           -- 窗口的别名是 w
```

2) 排序（序号）

使用窗口排序，可以对排序后的数据进行处理。图 9-39 a)所示的原始数据中，主键值不是连续的，主键值只要不为空，不重复就能满足主键约束的要求。

现在想要加一个连续的序号（以主键升序排序），代码如下，结果如图 9-40 a)所示。

```
Select id, name, course, score,
       row_number() over w as '序号'
from student
       window w as (order by id);
```

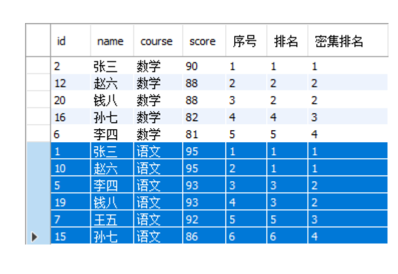
窗口中没有关键字 partition，表示所有数据作为一个分区，但是有 order by，表示这个窗口是排序的，这里是以 id 排序，所以序号是以 id 排序的。

id	name	course	score	序号
1	张三	语文	95	1
2	张三	数学	90	2
5	李四	语文	93	3
6	李四	数学	81	4
7	王五	语文	92	5
10	赵六	语文	95	6
12	赵六	数学	88	7
15	孙七	语文	86	8
16	孙七	数学	82	9
19	钱八	语文	93	10
20	钱八	数学	88	11

a) 以主键排序的序号

id	name	course	score	序号	排名	密集排名
1	张三	语文	95	1	1	1
10	赵六	语文	95	2	1	1
5	李四	语文	93	3	3	2
19	钱八	语文	93	4	3	2
7	王五	语文	92	5	5	3
2	张三	数学	90	6	6	4
12	赵六	数学	88	7	7	5
20	钱八	数学	88	8	7	5
15	孙七	语文	86	9	9	6
16	孙七	数学	82	10	10	7
6	李四	数学	81	11	11	8

b) 以成绩排序的序号、排名和密集排名



id	name	course	score	序号	排名	密集排名
2	张三	数学	90	1	1	1
12	赵六	数学	88	2	2	2
20	钱八	数学	88	3	2	2
16	孙七	数学	82	4	4	3
6	李四	数学	81	5	5	4
1	张三	语文	95	1	1	1
10	赵六	语文	95	2	1	1
5	李四	语文	93	3	3	2
19	钱八	语文	93	4	3	2
7	王五	语文	92	5	5	3
15	孙七	语文	86	6	6	4

c) 加上课程分区的序号、排名和密集排名

图 9-40 成绩表的序号、排名和密集排名

3) 排序（成绩排名）

如果要以成绩降序，以成绩从大到小进行排名，不仅列出序号，还要列出排名的名次，例如下述代码计算成绩的排名和密集排名，结果如图 9-40 b)所示。

```
Select id, name, course, score,
       row_number() over w as '序号',
       rank() over w as '排名',
       dense_rank() over w as '密集排名'
from student
       window w as (order by score desc);
```

其中窗口的别名 w 被 3 个窗口函数使用。从图 9-40 b)所示的结果很容易理解 rank()和 dense_rank()这两种排名方式的区别。

4) 分区和排序（分区的成绩排名）

如果加上课程名进行分区，再进行排名，代码如下，结果如图 9-40 c)所示。

```
Select id, name, course, score,
       row_number() over w as '序号',
       rank() over w as '排名',
       dense_rank() over w as '密集排名'
from student
       window w as (partition by course order by score desc);
```

9.5.4 合计与累计

在账务报表等各种报表中经常需要进行合计和累计。

1. 合计

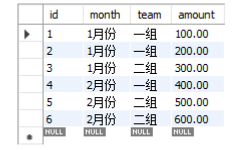
例如图 9-41 所示的是两个销售小组在两个月内的销售数据，创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”。实现合计功能的代码讲解如下。

1) 简单的合计——使用联合

如果要在原始数据的最后加上一行合计，可以通过联合来实现，代码如下，结果如图 9-42 所示。

```
Select * from sales
union
select 0, '合计', '', sum(amount) from sales;
```

2) 分组后的合计——使用联合



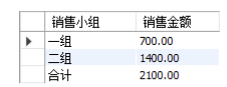
id	month	team	amount
1	1月份	一组	100.00
2	1月份	一组	200.00
3	1月份	二组	300.00
4	2月份	一组	400.00
5	2月份	二组	500.00
6	2月份	二组	600.00
NULL	NULL	NULL	NULL

图 9-41 原始销售数据



id	month	team	amount
1	1月份	一组	100.00
2	1月份	一组	200.00
3	1月份	二组	300.00
4	2月份	一组	400.00
5	2月份	二组	500.00
6	2月份	二组	600.00
0	合计		2100.00

图 9-42 加上合计行



销售小组	销售金额
一组	700.00
二组	1400.00
合计	2100.00

图 9-43 销售小组统计加合计

如果在要在销售小组分组统计的基础上，再加上合计行，代码如下，结果如图 9-43 所示。

```
Select team 销售小组, sum(amount) 销售金额 from sales
       group by team
union
select '合计', sum(amount) from sales;
```

3) 分组后的合计——使用 with rollup

使用 with rollup 关键字也可以实现上述代码的功能，代码如下，结果与图 9-43 完全相同。

```
Select coalesce(team, '合计') 销售小组, sum(amount) 销售金额 from sales
       group by team
              with rollup;
```

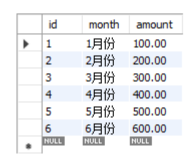
其中 coalesce 函数是当 team 列的值为 null 时，返回第二个参数的值。



对各种不同的需求，需要选择不同的技术或方法来实现。这个例子是非常典型的。

2. 累计

例如有一组每月销售数据，原始数据如图 9-44 所示，创建表和初始化数据的代码见附录 D “项目 3a 公共代码共享”。账务人员要求计算出当年每个月的销售金额以及累计销售金额，如图 9-45 所示。



id	month	amount
1	1月份	100.00
2	2月份	200.00
3	3月份	300.00
4	4月份	400.00
5	5月份	500.00
6	6月份	600.00
NULL	NULL	NULL

图 9-44 销售金额的原始数据

id	month	amount	累计
1	1月份	100.00	100.00
2	2月份	200.00	300.00
3	3月份	300.00	600.00
4	4月份	400.00	1000.00
5	5月份	500.00	1500.00
6	6月份	600.00	2100.00

图 9-45 销售金额的原始数据和累计金额

可以采用以下 4 种方法来实现相同的需求，这 4 种方法的运行结果与图 9-45 相同。

1) 方法一：临时变量法

使用一个临时变量保存累计金额，先初始化为 0，然后逐行累计，并为该变量起一个别名，用于显示。

```
Select id, month, amount,
        @cum := amount + @cum as  累计
    from sales, (select @cum := 0) as t
    order by id asc;
```

注意其中的赋值符是“:=”，而不是“=”，因为在 Select 语句中赋值必须用前者，在 Set 中赋值则两者都可以用。

2) 方法二：Join 法

用一个非等值的自连接，使用 sum()聚合函数计算小于等于当前行 id 的销售金额合计。

```
Select a.id, a.month, a.amount,
        sum(lt.amount)   as  累计
    from sales a join sales lt on a.id >= lt.id
    group by a.id
    order by id asc;
```

其中 on 的条件不是等于（=），而是大于等于（>=）。

3) 方法三：不等值连接法

这种方法与 Join 法非常相近，就是将非等值的自连接改为不等值的自连接。

```
Select a.id, a.month, a.amount,
        sum(lt.amount)   as  累计
    from sales a, sales lt
    where a.id >= lt.id
    group by a.id
    order by id asc;
```

其中 where 的条件不是等于（=），而是大于等于（>=）。

4) 方法四：窗口函数法

使用一个窗口和 sum()聚合函数进行累计，sum()函数即是聚合函数，又是一种窗口函数。

```
Select id, month, amount,
        sum(amount) over(order by id asc) as  累计
    from sales
    order by id asc;
```

上述代码还可写为如下，用 window 子句为窗口起一个别名。

```
Select id, month, amount,
        sum(amount) over(w) as  累计
    from sales
        window w as (order by id asc)
    order by id asc;
```

上述 4 种方法的结果是相同的，而窗口函数法是最为灵活的一种，也是性能较好的一种，建议尽量使用窗口函数。



对同一个需求，可以使用不同的技术或方法来实现，不同方法的可读性和性能有所差异。这个例子是非常典型的。

9.5.5 合并数据

在制作报表过程中，有时需要将多列或多行数据合并，例如图 9-46 所示的学生名单，创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”。下面以这组数据进行多列和多行数据的合并。

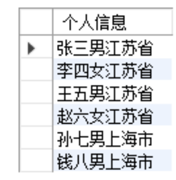
1. 合并多列数据——concat()函数

下述代码用 concat()函数将多列数据合并起来，查询结果如图 9-47 所示。

```
Select concat(name, sex, birth_place) 个人信息
       from student;
```

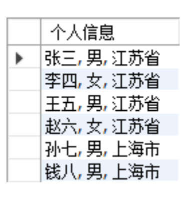
id	name	sex	birth_place
1	张三	男	江苏省
2	李四	女	江苏省
3	王五	男	江苏省
4	赵六	女	江苏省
5	孙七	男	上海市
6	钱八	男	上海市
NULL	NULL	NULL	NULL

图 9-46 原始数据



个人信息
张三男江苏省
李四女江苏省
王五男江苏省
赵六女江苏省
孙七男上海市
钱八男上海市

图 9-47 合并多列数据



个人信息
张三, 男, 江苏省
李四, 女, 江苏省
王五, 男, 江苏省
赵六, 女, 江苏省
孙七, 男, 上海市
钱八, 男, 上海市

图 9-48 合并多列数据（有分隔符）

2. 合并多列数据（有分隔符）——concat_ws()函数

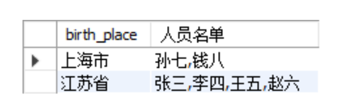
上述代码合并多列数据时，多列数据之间没有分隔，concat_ws()函数可以指定数据之间的分隔符，下述代码指定分隔符是“，”（第 1 个参数是分隔符），查询结果如图 9-48 所示。

```
Select concat_ws(', ',name, sex, birth_place) 个人信息
       from student;
```

3. 合并多行数据——group_concat()函数

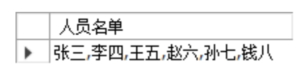
有时需要合并多行数据，这就要用到 group_concat()函数，该函数可以与 group by 子句合并使用，例如下述语句列出出生地为各省市的人员名单，查询结果如图 9-49 所示。

```
Select birth_place, group_concat(name) 人员名单
       from student
       group by birth_place;
```



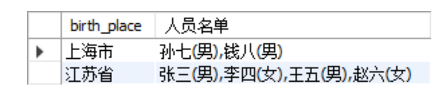
birth_place	人员名单
上海市	孙七,钱八
江苏省	张三,李四,王五,赵六

图 9-49 合并多行数据



人员名单
张三,李四,王五,赵六,孙七,钱八

图 9-50 合并所有数据



birth_place	人员名单
上海市	孙七(男),钱八(男)
江苏省	张三(男),李四(女),王五(男),赵六(女)

图 9-51 合并多行多列数据

如果没有 group by 子句，则代码如下，结果如图 9-50 所示。

```
Select group_concat(name) 人员名单
       from student;
```

4. 合并多行多列数据——group_concat()函数

还可以合并多行多列数据，例如下述代码，查询结果如图 9-51 所示。

```
Select birth_place, group_concat(name, '(', sex, ')') 人员名单
       from student
       group by birth_place;
```

9.5.6 动态 SQL 语句



附录 C 实训 9-4

在本书前面的讲解中，所有 SQL 语句都是事先编写好的。有时会需要根据用户的输入或表中的数据来编写 SQL 语句，这时就要用动态 SQL 语句，例如下述代码。

```
Use abc;
Drop table if exists employees;            -- 代码见附录 D “项目 3a 公共代码共享”
Create table employees(
        id int primary key not null auto_increment,
        name varchar(45) not null,
        department varchar(45)
);
Insert into employees values (1, '张三', '账务部'),(2, '李四', '销售部'),(3, '王五', '采购部');

Set @name = '张三';          -- 查询的姓名由用户输入，而不是程序员写的
Set @sql = concat('Select * from employees where name = ''', @name, ''';');
Prepare stmt from @sql;      -- 准备动态执行
Execute stmt;                -- 动态执行 SQL 文本中的语句
Deallocate prepare stmt;     -- 释放
```

上述代码中，变量@sql 的值是“Select * from employees where name = '张三'”，其中的“张三”不是事先编写的，而是动态添加到语句中的，然后执行这个变量中的 SQL 语句，得到需要的结果。

动态 SQL 语句比直接编写如下语句“Select * from employees where name = @name;”更加灵活，在有些场合下只有动态 SQL 语句一种解决办法，例如下一小节“行转列”的动态 SQL 语句应用实例。

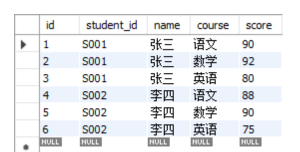
9.5.7 行转列与列转行

1. 行转列

在报表处理中，有时需要将行转为列，这里用 3 个例子进行讲解。

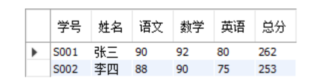
1) 已知关键行的数据

这里使用如图 9-52 所示的成绩表 score 为例进行讲解，创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”。将行转为列时，要根据学号 student_id 列的值，将同一学号的成绩转为不同的列，如图 9-53 所示。



id	student_id	name	course	score
1	S001	张三	语文	90
2	S001	张三	数学	92
3	S001	张三	英语	80
4	S002	李四	语文	88
5	S002	李四	数学	90
6	S002	李四	英语	75
NULL	NULL	NULL	NULL	NULL

图 9-52 成绩表的数据



学号	姓名	语文	数学	英语	总分
S001	张三	90	92	80	262
S002	李四	88	90	75	253

图 9-53 行转列后的成绩表

为此，以学号 student_id 和姓名 name 为分组列，统计每一门课程的成绩，把行中的数据转换为列中的数据，并且还可以计算总分，代码如下所示，运行结果如图 9-53 所示。

```
Select student_id 学号, name 姓名,
            sum(if(course='语文',score,0)) as '语文',
            sum(if(course='数学',score,0)) as '数学',
            sum(if(course='英语',score,0)) as '英语',
            sum(score) as '总分'
        from score
        group by student_id, name;
```

2) 关键行的数据需动态处理

上述例子是已知课程名称，并在代码中直接使用课程名称，如果课程名称是动态变化的，例如不同学生选修了不同的课程，这时就需要使用上一小节讲解的动态 SQL 语句。

为了演示不同的课程名称，用下述语句修改 id 为 3 的行的课程为“地理”。

```
Update score set course = '地理' where id = '3';
```

使用动态 SQL 语句的代码如下。

```
Set @sql0='';
```

```
Select @sql0 :=concat(@sql0,'sum(if(course= \''
            ,course
            ,'\',score,0)) as '
            ,course
            , ',') as sql0
      from (select distinct course from score
            ) a;

Set @sql = concat('Select student_id  学号, name  姓名, '
      ,@sql0
      ,' sum(score) as  总分  from score group by student_id, name;');

Select @sql;      -- 输入变量的值用于调试

Prepare stmt from @sql;
Execute stmt;
Deallocate prepare stmt;
```

上述代码中 select distinct course from score 的查询结果有 4 行，分别是语文、数学、地理和英语，其中地理是刚才更新的课程名。因此，变量@sql0 依次保存的值如图 9-54 所示。

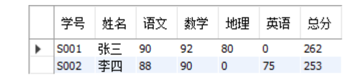
	sql0
▶	sum(if(course= '语文',score,0)) as 语文,
	sum(if(course= '语文',score,0)) as 语文,sum(if(course= '数学',score,0)) as 数学,
	sum(if(course= '语文',score,0)) as 语文,sum(if(course= '数学',score,0)) as 数学,sum(if(course= '地理',score,0)) as 地理,
	sum(if(course= '语文',score,0)) as 语文,sum(if(course= '数学',score,0)) as 数学,sum(if(course= '地理',score,0)) as 地理,sum(if(course= '英语',score,0)) as 英语,

图 9-54 变量@sql0 的值

变量@sql 保存的值如下，加粗斜体部分是变量@sql0 最后的值。

Select student_id 学号, name 姓名, ***sum(if(course= '语文',score,0)) as 语文,sum(if(course= '数学',score,0)) as 数学,sum(if(course= '地理',score,0)) as 地理,sum(if(course= '英语',score,0)) as 英语,*** sum(score) as 总分 from score group by student_id, name;   -- 调试输出的结果不完整，已根据前述代码补充完整

动态执行变量@sql 中的 SQL 语句，结果如图 9-55 所示。



	学号	姓名	语文	数学	地理	英语	总分
▶	S001	张三	90	92	80	0	262
	S002	李四	88	90	0	75	253

图 9-55 执行动态 SQL 语句（行转列）的结果

图 9-55 中的地理课程是因为上述对课程名称的更新而出现的，如果采用静态的语句就无法列出这门课程的成绩，或者任何课程的变化都要重写 SQL 语句，这在实际项目是无法实现的。

3) 行转列与 CTE

对于复杂的代码，可以巧妙地利用 CTE，使问题简化，编写出可读性好的语句。

例如使用如图 9-56 所示多种商品的销售数据，创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”。为方便处理，编写下述语句将原始数据转换为年度和季度的销售数据，处理后的数据如图 9-57 所示。

```
Select year(sale_date) y,                  -- y  表示年份
            ceil(month(sale_date)/3) m,    -- m  表示季度
            quantity,
            quantity*price amount
      from sales
```

id	sale_date	quantity	price
1	2023-09-05 14:37:25	5	89.90
2	2023-11-12 10:07:02	2	29.90
3	2023-11-25 08:17:07	3	39.99
4	2023-12-27 13:37:12	1	58.90
5	2024-01-07 11:32:26	3	18.80
6	2024-01-08 15:21:06	2	55.80
NULL	NULL	NULL	NULL

图 9-56 原始销售数据

y	m	quantity	amount
2023	3	5	449.50
2023	4	2	59.80
2023	4	3	119.97
2023	4	1	58.90
2024	1	3	56.40
2024	1	2	111.60

图 9-57 转换后的销售数据

然后将上述语句作为一个公用表表达式 CTE，编写行转列的代码，如下所示。

```
With sale as
	(select year(sale_date) y,
			ceil(month(sale_date)/3) m,
			quantity,
			quantity*price amount
		from sales
	)
select y  年份, sum(if(m=1,quantity,0)) 1 季度数量, sum(if(m=1,amount,0)) 1 季度金额
		, sum(if(m=2,quantity,0)) 2 季度数量, sum(if(m=2,amount,0)) 2 季度金额
		, sum(if(m=3,quantity,0)) 3 季度数量, sum(if(m=3,amount,0)) 3 季度金额
		, sum(if(m=4,quantity,0)) 4 季度数量, sum(if(m=4,amount,0)) 4 季度金额
	from sale
	group by y;
```

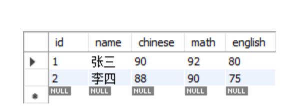
运行结果如图 9-58 所示，第一列是年份，其余 8 列分别是一年中 4 个季度的销售数量和金额，即使原始的销售数据有几百万行，运行的结果可以将多年销售数据汇总成一页，每年只有一行。

年份	1季度数量	1季度金额	2季度数量	2季度金额	3季度数量	3季度金额	4季度数量	4季度金额
2023	0	0.00	0	0.00	5	449.50	6	238.67
2024	5	168.00	0	0.00	0	0.00	0	0.00

图 9-58 按年度和季度统计的销售汇总

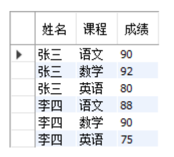
2. 列转行

列转行类似于行转列的逆操作，这里使用如图 9-59 所示的成绩表 score 数据，创建表和初始化数据的代码见附录 D“项目 3a 公共代码共享”。



id	name	chinese	math	english
1	张三	90	92	80
2	李四	88	90	75
NULL	NULL	NULL	NULL	NULL

图 9-59 列转行的演示数据



姓名	课程	成绩
张三	语文	90
张三	数学	92
张三	英语	80
李四	语文	88
李四	数学	90
李四	英语	75

图 9-60 列转行后的结果

将图 9-59 所示数据从列转为行，可以用联合查询来实现，代码如下，执行结果如图 9-60 所示。

```
Select name  姓名,'语文' as  课程, chinese as  成绩  from score
union all
select name,'数学', math from score
union all
select name,'英语', english from score
order by  姓名;
```

由于列是固定的，因此通常不需要用动态 SQL 语句来动态地处理列名。

9.6 性能优化与索引

数据库的性能与多个方面有关，包括计算机硬件（CPU、内存、外存、网络等）、软件（操作系统等）、以及数据库本身（数据结构、SQL 语句、索引、服务器配置和维护、数据分区和分表、负载均衡、以及监

控性能和调优等)。本小节主要讲解数据结构、SQL 语句的优化和索引的使用。

9.6.1 数据结构的优化

在设计表结构时，为提高数据库的性能，可以考虑如下几个因素。

- 表大小：尽量减少表的大小，表越大，占用的磁盘空间和需要的内存也越大，性能就越低。
- 列定义：尽量使用占用字节数少的数据类型，数字类比字符类的性能高，在字符类中，char 性能最高，varchar 次之，text 最低。列尽量不允许为空，非空列可以提高 Select 语句查询速度。
- 规范化设计：要求达到 3NF，使数据冗余减少到最小，只有在必要时，才会在局部使用反规范化设计手段，参见“2.5.3 规范化设计中的一些问题”。
- 索引：按照“6.3.2 索引的设计原则”来设计索引，只创建必要的索引。
- 存储引擎：根据需求选择合适的存储引擎，例如 MyISAM 引擎查询效率高，但是不支持外键约束和事务。

9.6.2 查询语句的优化

Select 语句担负了所有读取数据的任务，是影响性能的重要瓶颈，因此对 select 语句的优化要放在首要位置。

1. 避免列出所有列

查询所有列（select *…）有可能返回了不需要的列，应该指定具体的列名（select 列名列表…），只返回需要的列，不需要的列不返回给客户端。

2. 避免客户端分页

客户端分页需要返回全部结果，但是只显示结果集中当前页的 n 行，后续页的行经常被抛弃，而服务器端分页（使用 limit 子句分页）就可以避免这种情况，只返回需要显示的行，当需要下一页的行时，再向服务器请求下一页的内容。

3. 优化 join 操作

确保所有参与 join 的列都有相应的索引，这里指 on 条件中的列。

避免列出所有列，如果“select * from a join b on … join c on…”，这时将列出 a、b、c 三张表的所有列，而不是只列出所需要的部分列。

尽量减少 join 连接的表的数量，连接的表的数量越多，性能越差。反规范化设计就是因为要减少 join 的数量而在关键的部分故意保留了一定的冗余数据。

4. 优化 where、order by、group by 子句

这类优化通常需要对索引有更多的理解后，才能进行更好的优化，见下一小节的深入讲解。

9.6.3 索引的使用

前一小节讲解了查询语句的一般性优化，本小节先讲解 SQL 执行计划，了解查询的内部过程，再讲解 select 语句各个子句的优化。

测试用表的结构如下，这部分代码可从“项目 3a 公共代码共享”复制。

```
Create table user (
	id int primary key not null auto_increment,
	name varchar(45) not null,
	account varchar(45) default null,
	password varchar(45) default null
) engine=InnoDB;
```

测试数据库的性能需要表的行数足够大，可以用下述代码插入较多的行。

```
Drop procedure if exists p_insert_rows;
Delimiter $$
Create procedure p_insert_rows(row_count int)
Begin
Set @i = 0;
While @i < row_count do
	Set @i = @i + 1;
	-- 每循环一次插入一行随机数据
	Insert into user (name, account, password) values (md5(uuid()), md5(uuid()), md5(uuid()));
End while;
End$$
Delimiter ;

Call p_insert_rows(5000);
```

执行上述代码将插入 5000 行，多次调用该存储过程，使 user 表中包含大约 6 万条数据。其中的随机数据可以用下述函数产生。

- 通用唯一 ID：uuid()是一个长度为 36 的通用唯一 ID。
- 随机字符串：md5(uuid())是一个长度为 32 的随机字符串（根据唯一 ID 生成）。
- 随机整数：round(rand()*100, 0)是 0 到 100 之间的随机整数。
- 随机实数：round(rand()*100, 4)是 0 到 100 之间的 4 位小数的随机数字。

1. SQL 执行计划

1) Select 语句的执行过程

Select 语句的执行过程如下。

- 客户端向 MySQL 服务器发送一条查询请求。
- 服务器进行 SQL 解析、预处理、再由优化器生成对应的执行计划。
- MySQL 根据执行计划，调用存储引擎的 API 来执行查询。

对同一个查询目标，可以编写出不同的 select 语句，优化器可能会生成不同的执行计划，查询的性能也就不同。即使是对于完全相同的一条 select 语句，在不同的场景下（例如有无索引），优化器也会生成不同的执行计划，查询的性能也就不同。



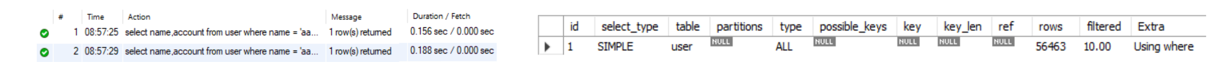
MySQL 8 之前的版本提供了查询缓存功能，执行查询时可能从缓存中取得查询结果。而 MySQL 8 取消了查询缓存。

2) 查看执行计划

查看执行计划是通过 explain 关键字模拟优化器执行 SQL 语句，查看 MySQL 是如何处理 SQL 语句的。例如下述代码在有 6 万行数据的 user 表中查找 name 为 aaa 的记录。

```
Select name,account from user where name = 'aaa';
```

在没有索引的情况下执行该语句两次，用时分别是 0.156 和 0.188 秒，如图 9-61 所示。



#	Time	Action	Message	Duration / Fetch
1	08:57:25	select name,account from user where name = 'aa...	1 row(s) returned	0.156 sec / 0.000 sec
2	08:57:29	select name,account from user where name = 'aa...	1 row(s) returned	0.188 sec / 0.000 sec

图 9-61 无索引时的查询用时

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	user	NULL	ALL	NULL	NULL	NULL	NULL	56463	10.00	Using where

图 9-62 无索引时的执行计划

在查询语句前加上 Explain，查看该语句的执行计划，结果如图 9-62 所示。

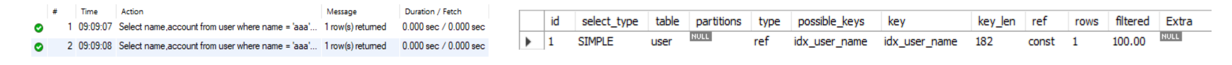
```
Explain Select name,account from user where name = 'aaa';
```

然后在 name 列上创建索引，代码如下。

```
Create index idx_user_name on user(name);
```

创建索引后再次执行上述查询两次，用时都是 0 秒，即用时小于 1 毫秒，如图 9-63 所示。同样的查询，使用索引后，性能提升了 100 多倍。

这时再用 Explain 查看该语句的执行计划，结果如图 9-64 所示。



#	Time	Action	Message	Duration / Fetch
1	09:09:07	Select name,account from user where name = 'aaa'...	1 row(s) returned	0.000 sec / 0.000 sec
2	09:09:08	Select name,account from user where name = 'aaa'...	1 row(s) returned	0.000 sec / 0.000 sec

图 9-63 有索引时的查询用时

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	user	NULL	ref	idx_user_name	idx_user_name	182	const	1	100.00	NULL

图 9-64 有索引时的执行计划

比较无索引和有索引时的执行计划，即比较图 9-62 和图 9-64，可以发现有一些不同，最明显的一个不同是 rows 列的数量，无索引时是 56463，而有索引时是 1，就是说，无索引时，需要读取 5 万多行才能找到目标行，而有索引时，只要读取 1 行就能找到目标行，这就是性能差别所在。最重要的差别是 type 列（连接类型），前者是 ALL（全表扫描），后者是 ref（使用非唯一索引）。

3) 执行计划结果中的行

图 9-62 和图 9-64 所示的执行计划都只有一行，如果查询是多表查询或子查询，则执行计划会有多行，每一行表示内部的一个 select 操作。这些行以 id 值进行标识（这里 id 没有主键的含义），id 值相同时表示是同一组查询，在有些情况下 id 值会递增，例如在子查询中。

4) 执行计划中的列

图 9-62 和图 9-64 所示的执行计划中主要的几个列的含义如下。

- id 列：已如前述，标识了一条查询语句内部的多个 select 操作。
- select_type 列：Select 类型，例如“simple”表示查询中不包含子查询或者 union，“primary”表示子查询中的父查询，“subquery”表示简单子查询中的子查询，“dependent subquery”表示相关子查询中的子查询，但是作为一张表出现在 from 子句中的子查询的类型则是“derived”，即派生表。
- table 列：Select 操作所涉及的表
- type 列：连接类型，即如何扫描表中的行，常见的 type 如表 9-1 所示。
- possible_keys 列：可能使用的索引，有时并不需要使用。为 NULL 则没有可能使用的索引。
- key 列：实际使用的索引。为 NULL 则没有实际使用索引。
- ref 列：索引与哪一列或常量 const 进行比较。
- rows 列：估计要扫描的行数，不是结果集的行数，也不是表的行数。
- Extra 列：附加信息，有许多种类的信息，例如“Using index”表示使用了覆盖索引，这时性能是很高的，而“Using where”表示使用了 where 条件，“Using temporary”表示查询还使用了临时表，这种查询性能很低。

表 9-1 常见的 type 类型（连接类型，性能由低到高排列）

连接类型	说明
ALL	全表扫描，从头到尾扫描整张表（表的每一行，因为没有索引才要全表扫描）。这是最慢的一种，性能最差
index	全索引扫描，和 ALL 类似，不同的是，index 扫描的是整个索引树，而不是整张表
range	索引列的范围扫描，例如 where 子句中有>、>=、<、<=、<>、is null、between、like、in 等运算符
ref	当满足索引的最左前缀规则，或者索引不是主键也不是唯一索引时使用
eq_ref	当使用了索引的全部组成部分，并且索引是 primary key 或 unique（并且该列是 not null）时使用
const	针对主键或唯一索引的等值查询扫描，最多只返回一行数据，性能非常好
system	表只有一行，是 const 的特例，性能最好

2. 覆盖查询与回表

在 name 列有索引的情况下，执行下述语句。

```
Select name, account from user where name = 'aaa';
```

当通过索引找到 name='aaa'的行时，还需要 account 列的数据，这时需要到表中根据主键值找到这一行的 account 列的数据，这就叫作回表。

对照下述语句。

```
Select id, name from user where name = 'aaa';
```

当通过索引找到 name = 'aaa'的行时，还需要的主键值在索引中就已经存在，不需要再到表中查询任何数据，因此称为覆盖查询，覆盖查询减少了一次回表读数据的操作，显然性能会更好。

当使用了覆盖索引时，执行计划显示 Extra 的信息是“Using index”，这种查询就叫覆盖查询。

3. 优化 count(*)

存储引擎对 count(*)查询的性能有极大的影响。下面对不同引擎进行对比，使用下述语句复制 user 表。

```
create table user1 like user;                 -- 复制表，除数据外，复制了表的一切，包括索引
alter table user1 engine=MyISAM;              -- 修改表的存储引擎，要在复制数据前修改，否则性能开销太大
insert into user1 select * from user;         -- 复制数据（所有行，共 60505 行）
```

这时新表 user1 使用 MyISAM 存储引擎，原表 user 是 InnoDB 的，两张表在 name 列上都有索引。

1) 存储引擎的影响

比较下述两行代码的执行用时。

```
Select count(*) from user;            -- 8.797 秒，user 的存储引擎是 InnoDB
Select count(*) from user1;           -- 0.000 秒，user1 的存储引擎是 MyISAM
```

差别达到上万倍，这是因为 InnoDB 引擎需要读取索引来计数，type 是 index，附加信息是 Using index，如图 9-65 所示。而 MyISAM 引擎对每一张表都保存了当前行数的值，查询 count(*)时直接读取该值，而不需要读取表或索引中的任何数据，附加信息是 Select tables optimized away，如图 9-66 所示。

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	user	NULL	index	NULL	idx_user_name	182	NULL	56463	100.00	Using index

图 9-65 InnoDB 的 count(*)执行计划

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	Select tables optimized away

图 9-66 MyISAM 的 count(*)执行计划

2) Where 条件的影响

对于 InnoDB 引擎，where 条件对 count(*)的影响比较大，比较下述两行代码的执行用时。

```
Select count(*) from user;                    -- 8.782 秒
Select count(*) from user where id > 0;       -- 0.188 秒
```

差别达到数十倍，查看它们的执行计划，前者的 type 是 index，后者的 type 是 range，range 的性能比 index 的性能好，因此相比之下后者的性能更好。

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	user	NULL	index	NULL	idx_user_name	182	NULL	56463	100.00	Using index

图 9-67 无 where 条件的 count(*)执行计划

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	user	NULL	range	PRIMARY,idx_user_name	PRIMARY	4	NULL	28231	100.00	Using where; Using inde

图 9-68 有 where 条件的 count(*)执行计划

4. 优化 Where

尽量避免在 where 子句中使用函数或表达式操作，它们会导致索引失效，也要避免在 where 子句中对列进行隐式类型转换，也会导致索引失效。

例如下述两条语句的结果是相同的。

```
select * from user where id + 1 = 50000;      -- 0.109 秒
select * from user where id = 49999;          -- 0.000 秒
```

第一条语句的 where 子句中使用了表达式，而无法使用主键的索引，需要对每一行都进行一次表达式求值，第二条语句则可以通过索引直接找到该行，二者执行时间相差超过百倍，通过执行计划可以发

现，前者的 type 类型是 ALL（全表扫描），后者的 type 类型是 const。

另外，要尽量简化 where 子句中的条件表达式，将非必需的条件从 where 子句移除。

5. 优化 Limit

下面这种优化法有一个条件，即要求数据是连续的，即没有 where 查询条件。例如下述代码，查询第 1000 页的数据。

```
Select * from user limit 49950, 50;        -- 每页 50 行，显示第 1000 页，(1000-1)*50 = 49950
```

优化为如下代码，先用子查询来查询得到第 1000 页的第一行数据，然后在父查询取其后的 50 行。

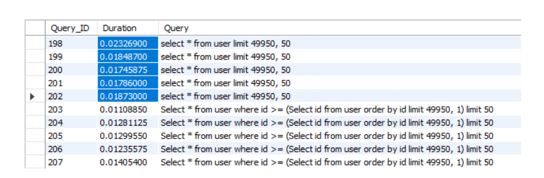
```
Select * from user where id >=
        (Select id from user order by id limit 49950, 1
        ) limit 50;
```

每条语句执行 5 次，平均执行时间分别是 0.019 秒和 0.013 秒，后者性能有所提升，当页数越大，性能提升就越明显。

这个例子的执行时间差别不大，所以采用了 5 次执行时间的平均值来进行比较，还可以通过 profiles 性能分析来获得执行时间，相关代码如下。

```
show variables like 'profiling';      -- 查看是否启用 profiles
set profiling=1;                      -- 启用，启用后再执行查询语句
show profiles;                        -- 查看 profiles 性能分析结果
```

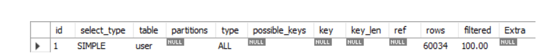
对前述两条语句的 Profiles 性能分析结果如图 9-69 所示。



Query_ID	Duration	Query
198	0.02326900	select * from user limit 49950, 50
199	0.01848700	select * from user limit 49950, 50
200	0.01745875	select * from user limit 49950, 50
201	0.01786000	select * from user limit 49950, 50
202	0.01873000	select * from user limit 49950, 50
203	0.01108850	Select * from user where id >= (Select id from user order by id limit 49950, 1) limit 50
204	0.01281125	Select * from user where id >= (Select id from user order by id limit 49950, 1) limit 50
205	0.01299550	Select * from user where id >= (Select id from user order by id limit 49950, 1) limit 50
206	0.01235575	Select * from user where id >= (Select id from user order by id limit 49950, 1) limit 50
207	0.01405400	Select * from user where id >= (Select id from user order by id limit 49950, 1) limit 50

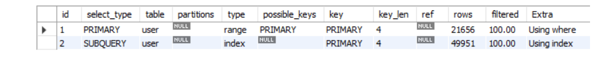
图 9-69 profiles 性能分析的结果

这两条语句的执行计划分别如图 9-70 和图 9-71 所示，前者的 type 是 ALL，后者虽然包含两个查询，但 type 分别是 range 和 index，子查询还使用了覆盖查询，因此总体性能有所提升。



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	user	NULL	ALL	NULL	NULL	NULL	NULL	60034	100.00	NULL

图 9-70 优化前的执行计划



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	user	NULL	range	PRIMARY	PRIMARY	4	NULL	21656	100.00	Using where
2	SUBQUERY	user	NULL	index	NULL	PRIMARY	4	NULL	49951	100.00	Using index

图 9-71 优化后的执行计划

6. 优化子查询

这里以单元 6 如图 6-12 所示的查询为例进行讲解。该查询是一个相关子查询，代码如下。

```
Select *,
                (Select col_name
                        from publisher where id=book.id_publisher
                ) 出版社
        from book;
```

也可以用连接查询实现相同的需求，代码如下。

```
Select book.*, publisher.col_name  出版社
        from book join publisher
                on book.id_publisher = publisher.id;
```

这两条语句的查询结果完全相同，由于数据量小，在执行时间上几乎没有差别，现在比较这两条语句的执行计划，分别如图 9-72 和图 9-73 所示。



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	book	NULL	ALL	NULL	NULL	NULL	NULL	9	100.00	NULL
2	DEPENDENT SUBQUERY	publisher	NULL	eq_ref	PRIMARY	PRIMARY	4	abc.b...	1	100.00	NULL

图 9-72 子查询的执行计划

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	publisher	NULL	index	PRIMARY	col_name	202	NULL	3	100.00	Using index
1	SIMPLE	book	NULL	ALL	NULL	NULL	NULL	NULL	9	11.11	Using where;...

图 9-73 连接查询的执行计划

一般来说，select_type 为 dependent subquery 的查询性能较低，就是说，相关子查询的性能较低，如果能改为连接查询，性能会得到提升。

另外，观察上述两个执行计划的 id 列，图 9-72 是一个子查询，id 分别是 1 和 2，图 9-73 是一个连接查询，id 都是 1。

9.6.4 慢查询分析

慢查询日志的目的是发现影响性能的 SQL 语句，并进行分析和优化。

慢查询最常见的原因是索引使用不当，特别是表的行数增加时，会显著降低查询语句的性能。通常在项目开始时，性能表现还比较好，随着项目运行时间越来越长，表的行数越来越多，性能问题会逐渐突显，作为数据库的管理员，要经常关心慢查询日志中的记录，可以从下述几个方面进行优化。

- 查看慢查询语句的执行计划，分析是否有优化的可能。
- 为 type 为 ALL 的查询创建索引，有时需要删除多余的索引。
- 其他方面的优化，如硬件、服务器参数、体系架构、分区或分表、数据结构等。

9.7 安全技巧

在 MySQL 上的账号密码可能有两套系统，一是单元 8“8.3 数据库安全”讲解的用户和授权，是 MySQL 内置的安全措施，二是每个应用项目都可能会有一套自己的账号系统，例如实战演练平台上的每个项目都可以使用自己的账号系统，只有通过登录进入系统，才能使用该项目。在单元 8“【案例讲解】应用项目的管理”已经作过详细讲解。

开发者在设计一个项目时，需要考虑到一些安全因素。本小节主要讲解应用项目本身的账号系统中，对密码的安全保护。

9.7.1 防止 SQL 注入攻击

SQL 注入攻击的原理是，如果用户的密码是明码保存，例如用于验证用户登录身份的信息如图 9-74 所示。



```
mysql> Select * from user limit 1;
+----+------+----------+----------+
| id | name | account  | password |
+----+------+----------+----------+
|  1 | 张三 | Zhangsan | dfgd34df |
+----+------+----------+----------+
1 row in set (0.00 sec)
```

图 9-74 用户登录信息

如果用户身份认证的代码采用如下代码。

```
Select * from user where account = 'Zhangsan' and password = 'dfgd34df';
```

上述语句比对账号和密码，两者正确时，用户登录成功。

这种设计给了攻击者一个可乘之机，攻击者使用万能密码“abc' or 1=1 limit 1; -- x”，注入到 SQL 语句中，此时无论登录时使用什么账号，注入的万能密码使 where 条件变为永远成立，如下所示。

```
Select * from user where account = 'xyz' and password = 'abc' or 1=1 limit 1; -- x';
```

在这个万能密码“abc' or 1=1 limit 1; -- x”中，abc 可以是任意字符，单引号结束密码字符串，核心部分是 or 1=1，因为这会使整个条件表达式为真，limit 1 保证了只返回一行，不论账号是什么，总是以第一

行的用户身份登录，注释部分把原语句中的最后部分一起注释掉。



在实战演练平台上打开“项目 9d SQL 注入攻击”，尝试用万能密码登录，这时应能成功登录，退出后再尝试用正确的账号和密码登录，比较两次登录中，运行日志记录下来的 SQL 语句，复制到 MySQL Workbench 中运行，从而理解 SQL 注入攻击。

在项目 9d 中用上述万能密码登录，不论账号是什么，总是以“张三”的身登录，如果在表中添加一个用户，要登录为这个用户身份，万能密码要怎么改？

9.7.2 密码的加密和加盐

密码的明码保存有两个可能的安全隐患，一是如前所述的 SQL 注入攻击，二是密码泄露，如果黑客通过某种方式盗取到了数据库中的数据，那么以明码保存的数以万计的用户密码将全部被黑客掌握，另外，系统管理员作为权限最高的内部人员，也能够轻而易举的获得任何一个用户的密码。

对于明码保存的密码，可以采用 md5 算法进行加密密，只需作极少的修改，代码如下。

```
Insert into user (id, name, account, password) values (null, '李四', 'Lisi', md5('dfgd34df'));
```

插入的数据如图 9-75 所示，原始密码“dfgd34df”加密后成为“9d6532e926db1a90b94ea82d085350b7”，md5 是一种不可逆的加密算法，只能加密，无法解密。



mysql> select * from user;

id	name	account	password
1	张三	Zhangsan	dfgd34df
2	李四	Lisi	9d6532e926db1a90b94ea82d085350b7

2 rows in set (0.00 sec)

图 9-75 明码密码和加密密码的比较

登录用的查询语句应该改写为如下。

```
Select * from user where account = 'Lisi' and password = md5('dfgd34df');
```



在实战演练平台上打开“项目 9e 密码的加密”，尝试上一小节的万能密码登录，这时万能密码失效，因为项目 9e 将用户密码改为加密码保存，即使是管理员也无法看到用户的密码。比较一下项目 9d 和项目 9e 的区别，主要体现在以下两处。

- 项目 9e 在用户管理功能的“保存数据 SQL 语句”，加上了对密码进行 md5 加密的语句。
- 项目 9e 在“开发工具”的“通用全局设置”的身份认证代码，使用 md5 函数后再比对密码。

分别在这两个项目中查看运行日志，比较两个项目的登录用 SQL 语句，就能更好地理解它们的区别。或者切换到开发版后以系统管理员身份登录，查看登录用 SQL 语句是如何编写的，参见单元 8“【案例讲解】应用项目的管理”。

为了进一步加强密码的安全，还可以采用加盐技术，进一步加强密码的安全保护。例如单元 8 图 8-3 所示的密码是加盐加密保存的。

9.7.3 加密传输

在网络上传输数据，如果采用 http 协议，传输的是明码，用户登录时提交的账号和密码都是明码传输的，黑客可以在传输的途中截获账号和密码，从而攻击相应的用户账号。

如果采用 https 协议（其中的 s 是 secure 的首字母，意为安全），数据是加密传输的，黑客既使截获到数据，也是加密的，并且是与其他数据一起加密，无法分离出账号和密码，更无法破解。

因此在访问网站时，如果地址是 http://起头的，那么是不安全的，如果地址是 https://起头的，那么是比较安全的，所以所有重要的网站（如银行的网站），必定是使用 https 协议。

9.7.4 密码安全策略

常用的密码设置策略有如下几种，可以强制性要求用户遵守这些规则，这在各种 APP 中十分常见。

- 密码的最小长度，如最短 8 个字符。
- 密码的复杂度，如必须含有大写、小写字母、数字、符号的组合。
- 密码的定时更换，例如每 2 个月必须更换一次。

密码设置策略越复杂，对用户的友好度越低，因此应该选择合理的密码设置策略。

【案例讲解】进销存管理系统



打开附录 D“项目 9f 进销存管理系统的开发”，在实战演练平台上体验进销存管理系统的演示版，以超级用户身份登录（super/super），如图 9-76 所示。



图 9-76 进销存管理系统

任务 1 探索数据结构

在运行项目之前，需要理解项目的数据结构，从以下几个方面来查看数据结构。

- 通过建模工具的逆向工程，获得数据结构的 EER 图。参见“3.5.6 正向工程和逆向工程”。
- 在 MySQL Workbench 中查看数据库中表的数据结构。
- 在实战演练平台“开发工具”的“列出表的结构”中，直接查看表的结构，或者先提取表结构，然后将提取的结果复制到 Excel 中，形成开发文档。如图 9-77 所示。



图 9-77 在实战演练平台中查看数据结构

任务 2 体验项目功能

运行项目，阅读演示版中的项目介绍和使用说明，了解项目的流程，理解项目的实施。相关说明见如图 9-76 所示的下方。

任务 3 探索代码并改进或开发新功能

切换到开发版，探索每一个功能的 SQL 代码，修改或添加功能。有兴趣的读者可以参照演示版，开发一个自己的进销存管理系统。

【实战演练】自选题目

可以采用以下两种方式进行实战演练。

- 从附录 D 提供的实战项目中选择一个项目，切换到开发版，对项目作进一步的开发，探索项目的 SQL 代码，并进行一些修改或增加功能。
- 自行选择一个题目，可难可易，在实战演练平台上完成项目，也可以参考一些书籍提供的教学案例，在实战演练平台上完成案例项目的开发。

任务 1 如何选题

读者可以根据自己的兴趣和对业务的熟悉情况来选择题目，也可以在选择题目后，再到市场中进行考察调研，例如选择“房屋出租管理系统”后，可以以租客的身份到房屋中介公司，通过租房咨询来了解房屋出租的流程和填写的数据，在与业务员的沟通中了解用户的需求。

这里列出一些选题：学生成绩管理系统、学生选课管理系统、毕业设计选题管理系统、学校教材订购系统、学生宿舍管理系统、学生社团管理系统、事业单位办公管理系统、人力资源管理系统、人才招聘管理系统、小区物业管理系统、社区服务管理系统、房屋出租管理系统、企业设备管理系统和科研项目管理系统等等。

开发全新项目时需要采用如下格式的网址（每个数据库名和子项目标识的组合就是一个新项目）。

```
http://127.0.0.1:8090/mysql/数据库名?app=子项目标识
```

如果数据库名为空，表示使用默认数据库 mysqlv2，如果子项目标识为空，表示没有标识。

任务 2 开发流程

开发过程如下所示。

- 项目选题：根据个人爱好，以及对项目业务的熟悉程度进行选题。
- 需求分析：参考单元 2 中与需求分析有关的部分，进行数据分析和功能设计。
- 数据结构设计：参考单元 2 中与规范化设计有关部分，进行数据结构设计。
- 创建数据库：数据库名称是网址中的数据库名，同一个数据库可以有多个子项目。
- 创建表：建议使用开发版中“开发工具”的“创建表或修改表”工具。
- 功能开发：只需要编写 SQL 语句就能完成项目的开发，不需要任何其他语言的语句。
- 测试运行：通过测试运行，发现 bug，进行修改和进一步开发。

【单元重点】

单元小结参见本单元的【思维导图】，单元重点如下。

- 使用实战演练平台开发项目
- 实战技巧（视图的应用、事务的应用、查询技巧、性能优化和安全技巧）
- 项目开发的全过程，包括需求分析、数据结构设计、开发实施、运行维护等各个阶段。

【课后思考】

1. 为什么说需求分析是项目成败的关键环节？
2. 数据结构设计与需求分析有什么关系？

【课外拓展】

1、 问一问 AI，有关密码加盐技术的问题，了解它的原理和作用。

2、 从网上查找名为“阿里巴巴 Java 开发手册-2022.pdf”的文件，阅读你感兴趣的部分。

附录 A MySQL 常用数据类型

MySQL 常用数据类型，如附表 1 所示。

附表 1・MySQL 常用数据类型

分类	数据类型	字节数	含义	说明（范围）
整数	TinyInt	1	微整数	有符号时：-128~127，无符号时：0~255
	SmallInt	2	短整数	有符号时：-32768~32767，无符号时：0~65535
	MediumInt	3	中整数	有符号时：-8388608~8388607，无符号时：0~16777215
	Int	4	整数	有符号时：大约-20 亿~20 亿，无符号时：大约 0~40 亿
	BigInt	8	长整数	有符号时：大约-900 亿亿~900 亿亿
浮点数	Float[(M,D)]	4	单精度浮点数	可选：M 显示的宽度，D 显示的小数位数
	Double[(M,D)]	8	双精度浮点数	可选：M 显示的宽度，D 显示的小数位数
	Decimal (P,D)	可变	精确浮点数	必选：P 精度（1~65），D 小数位数（0~30），要求 D <= P。
字符串	Char(n)	n	定长字符串	n 取值范围：0-255，占用 n 字节，不论实际长度是多少
	Varchar(n)	数据长度	变长字符串	n 取值范围：0-65535，占用实际长度
	Tinytext	数据长度	短文本	0-255 字节，短文本数据
	Text	数据长度	文本	0-65 535 字节，文本数据
	Mediumtext	数据长度	长文本	0-16 777 215 字节，长文本数据
	Longtext	数据长度	极长文本	0-4 294 967 295 字节，极长文本数据
二进制	Binary(n)	n	定长二进制	n 取值范围：0-255，占用 n 字节，不论实际长度是多少
	Varbinary(n)	数据长度	变长二进制	n 取值范围：0-65535，占用实际长度
	Tinyblob	数据长度	短二进制	0-255 字节，短二进制数据
	Blob	数据长度	二进制	0-65 535 字节，二进制数据
	Mediumblob	数据长度	长二进制	0-16 777 215 字节，长二进制数据
	Longblob	数据长度	极长二进制	0-4 294 967 295 字节，极长二进制数据
日期时间	Date	3	日期	'1000-01-01' 到 '9999-12-31'
	Time	3	时间	'-838:59:59' 到 '838:59:59'
	Year	1	年份	1901 到 2155
	Datetime	8	日期和时间	'1000-01-01 00:00:00' 到 '9999-12-31 23:59:59'
	Timestamp	4	时间戳，含时区	'1970-01-01 00:00:00' 到 '2038-01-19 03:14:07'，含时区

注：对于变长的文本类型和二进制类型，占用空间等于数据长度加上内部开销。

附录 B MySQL 常用内置函数

MySQL 常用内置函数，如附表 2 所示。

附表 2 MySQL 常用内置函数

分类	函数名	功能
聚合函数	avg(expression)	返回表达式中各值的平均值，忽略 null 值
	sum(expression)	返回表达式中所有值的和，只用于数字列
	min(expression)	返回表达式的最小值
	max(expression)	返回表达式的最大值
	count(expression)	返回结果的行数，忽略 expression 为 null 值的行
	count(*)	返回结果的行数，包括 null 值
数学函数	abs(numeric)	返回指定数值表达式的绝对值（正值）
	sqrt(float)	返回指定浮点值的平方根
	pow(float, y)	返回指定表达式的指定幂的值
	exp(float)	返回指定的 float 表达式的指数值
	log(float)	返回指定 float 表达式的自然对数
	log10(float)	返回指定 float 表达式的常用对数（以 10 为底）
	floor(numeric)	返回小于或等于指定数值表达式的最大整数
	ceil(numeric)	返回大于或等于指定数值表达式的最小整数
	round(numeric, length[, function])	返回一个数值，舍入到指定的长度或精度
	rand([seed])	返回介于 0 到 1 之间的伪随机 float 值
	sin(float)	三角函数（正弦、余弦等）
字符串函数	length(string)	返回指定字符串表达式的字节数（占用空间，汉字占用多个字节）
	char_length(string)	返回指定字符串表达式的字符数（实际字符，汉字为 1 字符）
	substring(expression, start, length)	返回字符、二进制、文本或图像表达式的一部分
	left(character, integer)	返回字符串中从左边开始指定个数的字符
	right(character, integer)	返回字符串中从右边开始指定个数的字符
	replace(string, pattern, replacement)	用另一个字符串值替换出现的所有指定字符串值
	concat(string1, string2, …)	返回多个字符串连接而成的字符串，注意：不能用加号连接字符串
	ascii(character)	返回字符表达式中最左侧的字符的 ascii 代码值
	char(integer)	将 int ascii 代码转换为字符
	ltrim(character)	返回删除了前导空格之后的字符表达式
	rtrim(character)	截断所有尾随空格后，返回一个字符串
	upper(character)	返回大写的字符表达式
	lower(character)	返回小写的字符表达式
日	curdate()	返回当前日期

期和时间函数	curtime()	返回当前时间
	now()	返回当前日期时间
	day(date)	返回表示指定 date 的“日”部分的整数
	month(date)	返回表示指定 date 的“月”部分的整数
	year(date)	返回表示指定 date 的“年”部分的整数
	datediff(enddate, startdate)	返回两个指定日期之间相差的天数
	adddate(date, interval number day)	将一个日期时间加上指定的间隔（天、月或其他）
	subdate(date, interval number day)	将一个日期时间减去指定的间隔（天、月或其他）
系统函数	version()	返回数据库的版本号
	database()	返回当前打开的数据库名
	user()	返回当前登录的用户名
	last_insert_id()	在 Insert 语句之后，返回最新插入的行的主键值（自动生成的）
	found_rows()	Select 语句查询到的行数，或者 update 语句匹配的行数
	row_count()	在 Insert、update、delete 语句之后，返回实际影响的行数
转换函数	cast(expression as data_type[(length)])	将表达式的值转换为另一种数据类型，是 ANSI 标准
	convert(expression, data_type[(length)])	将表达式的值转换为另一种数据类型，功能与 cast 完全相同

附录 C Jitor 实训列表

Jitor 实训是一种在线实训，需要实时联网，可以对学生编写的代码和运行的结果进行实时评价，教师可以实时监测全班学生的实训进展情况。

本书提供 40 余个 Jitor 在线实训，如附表 3 所示，在引用处有如右图的标识。使用说明见附表 3 之后的说明。



附录 C
实训 1-1

附表 3 实训列表

序号	实训名称	序号	实训名称
1	【实训 1-1】“Hello，数据库”项目	23	【实训 6-1】简单子查询
2	【实训 1-2】“Goods 数据库”项目	24	【实训 6-2】相关子查询
3	【实训 2-1】“图书信息数据库”项目	25	【实训 6-3】增删改与子查询
4	【实训 3-1】创建表	26	【实训 6-4】视图
5	【实训 3-2】创建主表和从表	27	【实训 6-5】索引
6	【实训 3-3】变更表	28	【实训 7-1】MySQL 编程基础
7	【实训 3-4】添加外键约束	29	【实训 7-2】流程控制语句
8	【实训 3-5】创建书店管理数据库	30	【实训 7-3】内置函数
9	【实训 4-1】数据插入	31	【实训 7-4】存储函数
10	【实训 4-2】数据更新	32	【实训 7-5】存储过程
11	【实训 4-3】数据删除和清空	33	【实训 7-6】触发器
12	【实训 4-4】数据操纵与数据约束	34	【实训 7-7】事件
13	【实训 5-1】单表查询——选择列	35	【实训 7-8】事务和锁
14	【实训 5-2】单表查询——Where 子句	36	【实训 8-1】MySQL 命令行
15	【实训 5-3】单表查询——Order by 子句	37	【实训 8-2】MySQL 服务器
16	【实训 5-4】单表查询——Limit 子句	38	【实训 8-3】数据库安全
17	【实训 5-5】联合查询	39	【实训 8-4】数据库备份与恢复
18	【实训 5-6】连接查询——两张表的连接	40	【实训 8-5】数据库日志
19	【实训 5-7】连接查询——内连接查询	41	【实训 9-1】公用表表达式 CTE
20	【实训 5-8】连接查询——外连接查询	42	【实训 9-2】临时表
21	【实训 5-9】连接查询——自连接查询	43	【实训 9-3】窗口和窗口函数
22	【实训 5-10】聚合查询	44	【实训 9-4】动态 SQL 语句

使用说明

实训的用途比较广泛，可作为课堂讲授时的演示（按步骤讲解，方便复制代码），作为课中的课堂练习或课后的作业，也可作为随堂小测或考试使用。

1. 教师使用“Jito 教师端”网站，使用步骤如下。

- 访问管理网站：“Jitor 实训教学平台”的网址是 http://jit.ngweb.org:8092/jitor/。
- 为学生创建访问账号：先创建教学班级，然后批量导入学生学号和姓名。

- 安排实训内容：为教学班级选择要做的实训，指定开始时间和结束时间。
- 检查实训进度和得分：实时查看学生的实训进度，了解学生的进度，发现差生。
- 下载成绩：在学期结束时，下载学生所有成绩，包括学期总评成绩。

2. 学生使用“Jitor 学生端”软件（称为“Jitor 校验器”），使用步骤如下。

1) 安装启动“Jitor 校验器”

- 下载解压：从本书附录 E 提供的地址下载“Jitor 校验器”（http://ngweb.org/mysql/?p=8），文件名“jitor-v6.2.zip”，然后解压到本地硬盘的根目录下，目录名是 jitor-v6.2。
- 启动：双击解压目录中的“Jitor 启动.bat”启动它。
- 配置密码：配置访问 MySQL 的密码，即系统管理员 root 的密码，方法是在 Jitor 校验器上点击“配置”→“连接测试”，打开“JDBC 连接测试”窗口，测试通过后，自动保存密码。

2) 使用 Jitor 实训

- 登录：作为学生，使用教师提供的账号密码登录。作为普通读者，自行注册一个账号后登录。
- 打开实训：从实训列表中打开教师安排的实训。
- 完成实训：按照实训内容的要求操作，每做完一步，成功通过后再做下一步，直到完成。
- 查看得分：随时查看实训的得分情况，成功通过得 7 分，每失败一次扣 1 分。

3) 体验 Jitor 实训

如果只是想体验 Jitor 实训，可以无需注册账号，在 Jitor 校验器中用账号 mysqlb-test 和密码 123456 直接登录，体验本书所有实训，如果密码错（有人修改了密码），请等下一个小时重试，该账号的密码每小时整点恢复一次，实训记录每日凌晨清除一次。

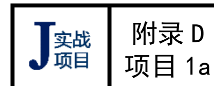
4) 使用“实战演示平台”

Jitor 校验器同时也是“实战演示平台”的 Web 服务器，为其提供后台服务，访问本地 MySQL 服务器，在成功配置密码后，通过 http://ngweb.org/mysql/?p=3 上的链接打开附录 D 列出的所有实战项目。

附录 D 实战项目列表

实战项目由两部分组成：项目案例 + 项目网站，其中的项目网站需要 Jitor 校验器作为 Web 服务器，读者不需要编写任何其他语言的代码，仅用 select 一条语句开发一个具有增删改查功能的网站。仅在需要定制功能时，才要编写 insert、update、delete 和事务处理语句。

本书提供 10 余个实战项目（项目案例+项目网站），如附表 4 所示，在引用处有如右图的标识。网址是 http://ngweb.org/mysql/?p=3，使用说明见附表 4 之后的说明。



附表 4 实战项目列表

序号	单元	实战项目名称	数据库	app 标识	项目用途
1	单元 1 认识数据库（1.3.3）	项目 1a “Hello，数据库”项目	book	b1a	理解主键
2	单元 2 理解关系数据库（2.2.3）	项目 2a “图书信息数据库”项目	bookinfo	b2a	理解外键
3	单元 2 理解关系数据库（2.3.4）	项目 2b “学生成绩管理”项目	student	b2b	理解多对多
4	单元 2～单元 7【案例讲解】	项目 2c 书店管理项目	mybook	b2c	案例讲解
5	单元 2～单元 8【实战演练】	项目 2d 图书借阅项目	mybook	b2d	实战演练
6	单元 3～单元 9【代码共享】	项目 3a 公共代码共享	mysqlv2	b3a	代码共享
7	单元 8 数据库管理【案例讲解】	项目 8a 应用项目的管理	mybook	b8a	功能演示
8	单元 9 实战项目（9.1）	项目 9a 实战演练平台功能演示	mybook	b9a	功能演示
9	单元 9 实战项目（9.2）	项目 9b 图书信息数据库的开发	mybook	b9b	开发操作
10	单元 9 实战项目（9.3）	项目 9c 视图的应用和事务的应用	mybook	b9c	知识点讲解
11	单元 9 实战项目（9.6.1）	项目 9d SQL 注入攻击	mybook	b9d	知识点讲解
12	单元 9 实战项目（9.6.2）	项目 9e 密码的加密	mybook	b9e	知识点讲解
13	单元 9 实战项目【案例讲解】	项目 9f 进销存管理系统的开发	jxc	b9f	案例讲解
14	单元 9 实战项目【实战演练】	自选题目	自选	自选	实战演练

使用说明

1. 使用方法

- 启动 Jitor 校验器：安装方法见附录 C，启动后作为 Web 服务器，保持运行状态，不要关闭。
- 打开项目：访问 http://ngweb.org/mysql/?p=3，从该网页的链接地址打开项目。

如有疑问，参见单元 9 的“9.1 实战演练平台的安装和使用”。

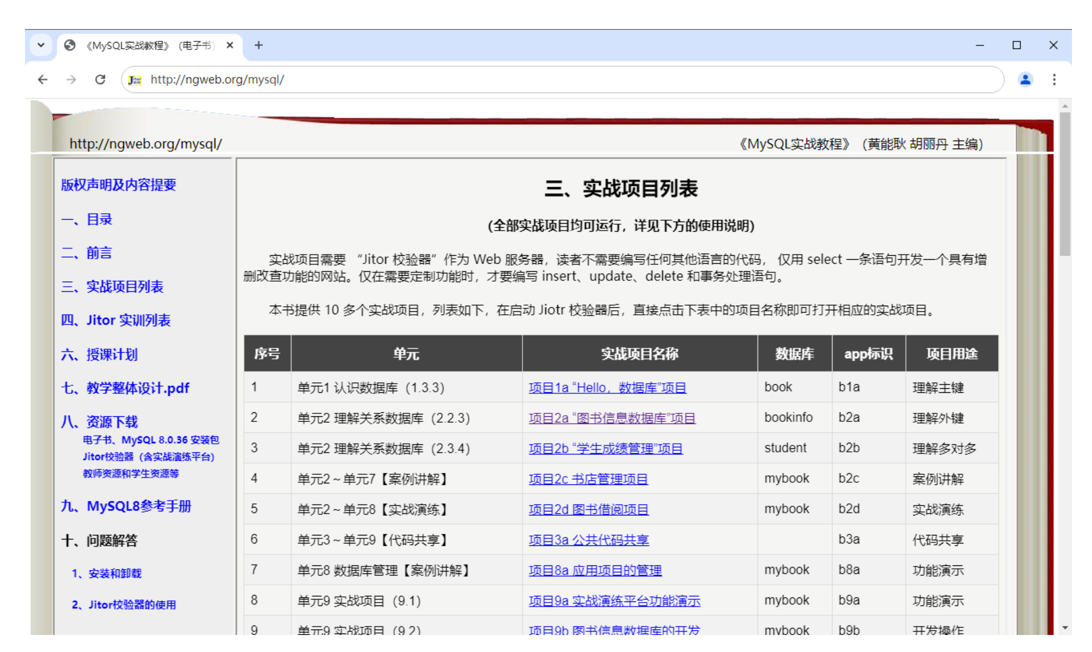
2. 实战项目的使用

实战项目的使用有如下 4 种形式。

- 只体验项目演示，以演示方式运行，目的是了解项目运行过程，查看运行日志的记录。
- 切换到开发版（在“帮助”中有链接），然后探索演示项目的 SQL 代码，并修改或增加功能。
- 切换到开发版，只利用演示项目的数据结构，重新开发一个项目。
- 自行选题，开发一个全新的项目，开发过程参见单元 9 的“【实战演练】自选题目”。

附录 E 在线资源说明

本附录在线提供实战项目列表、实训列表、授课计划和教学整体设计等，如附图 1 所示，还提供资源下载的网盘链接，如附图 2 所示。附图 1 所示的界面可以通过链接直接打开实战项目。



附图 1 在线资源界面



附图 2 资源下载的网盘内容

在线资源的访问地址是 http://ngweb.org/mysql/?p=8，或者扫描如附图 3 所示的二维码。

资源下载中还包括了一些延伸阅读资料，例如“单元 10 PHP 应用编程”，还包括全国二级等考用的 PHP 编程环境 wampserver 软件。



附图 3 在线资源二维码

==== 全书结束 ====